Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Grgič Jelen

Prepoznava tekačev na slikah

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Borut Batagelj

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj v svojem delu pregleda metode računalniškega vida s katerimi bi lahko na fotografiji prepoznal osebo. Tek postaja vedno bolj popularen med rekreativnimi športniki. S pomočjo najrazličnejših naprav za zajem fotografij in video posnetkov je na koncu tekaškega dogodka na voljo veliko slik. Ne teh slikah se običajno nahaja več oseb, ki se med seboj tudi zakrivajo. V svojem delu predlagajte postopek, ki bo na dani fotografiji samodejno prepoznal posamezne tekače. Preglejte metode za detekcijo obraza, telesa in številke, ki jo nosi posamezen tekač. Za predlagano metodo preverite tudi uspešnost na zbirki fotografij tekačev.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
OCR	Optical characted recognition	optična prepoznava znakov
API	application programming interface	aplikacijski programski vmesnik

Povzetek

Naslov: Prepoznava tekačev na slikah

Avtor: Miha Grgič Jelen

V diplomski nalogi smo razvili rešitev za identifikacijo tekačev na fotografijah športnih prireditev. Za dosego tega cilja, smo izdelali postopek, ki na fotografiji najprej izolira posamezne osebe, nato za vsako osebo izračuna tarčno regijo v kateri se najverjetneje nahaja štartna številka. V naslednjem koraku izvede segmentacijo tarčne regije in izolira posamezne števke. Nato vsak izsek, ki je potencialno števka poskušamo prepoznati z optičnem branjem znakov z uporabo orodja TesseractOCR. Na koncu prepoznane števke dodatno filtriramo in združimo v štartno številko. Predstavljena so vsa uporabljena orodja ter teorija in principi kako sploh delujejo. Nekoliko bolj podrobno smo se posvetili samim detektorjem objektov, saj je ta del ključen za uspešnost celotnega postopka. Opisan je tudi razvit uporabniški vmesnik, ki nam močno olajša pregledovanje fotografij in vizualizacijo rezultatov. Na koncu pregledamo nerešene probleme ter nekaj idej za nadaljni razvoj.
Ključne besede: detektor objektov, detektor oseb, prepoznava številk, optična prepoznava znakov, nevronske mreže, OpenCV, selektivno iskanje.

Abstract

In the thessis we have developed a solution for identifying runners from photos taken during sport events. For reaching that goal, we developed a procedure, which first isolates each person on the photo. Then we define a target region (for each person), where we belive that the number on the jersey is located. In the next step we execute further image segmentation in order to isolate single digits. After that we use optical character recognition (TesseractOCR) on each of the isolated digit candidates, to determine if it's a number. In the last steps, we perform further result filtering to remove false positive results. The rest of them is joined into the final number. All the tools we have used and the principles behind them, are presented in the paper. We were focusing on object detectors, since this is the key-point in the whole procedure. We also described the user interface we have created for navigation and result visualisation. In the end we explain problems we were not able to solve and ideas for further development.

Title: Runner's identification from the photos

Author: Miha Grgič Jelen

Keywords: object detector, person detector, number detection, optical character recognition, neuron networks, OpenCV, selective search.

Poglavje 1

Uvod

Kot študent sem več let delal kot fotograf tekaških prireditev v Sežani. Širši javnosti so bolj znana pod imeni Mali kraški maraton ter Tekaški pozdrav jeseni. Z voznikom sem na motorju vijugal med tekači in jih fotografiral. Nastale fotografije so bile uporabljene za promocijske ter medijske potrebe, večina pa jih je končala v spletni galeriji. Tam so bile tudi fotografije ostalih fotografov. Skoraj vsak udeleženec po prihodu domov pregleda svoje rezultate. V večini primerov pa si potem ogleda spletno galerijo, v upanju da bo našel tudi kakšno svojo fotografijo. V tem trenutku, pa je njegova vztrajnost na resnem preizkusu, saj je v spletni galeriji veliko število fotografij. V kolikor ima udeleženec srečo, najde svojo fotografijo med prvimi 100. Sicer pa ponavadi odneha veliko prej, kot pregleda vseh nekaj 100 fotografij iz galerije, v upanju, da se bo našel na kateri izmed njih.

Tako je nastala ideja o razvoju prototipa za računalniško identifikacijo oseb na fotografijah, z namenom da bi udeleženci prireditev lahko enostavno dostopali do svojih fotografij. Najprej smo pregledali, če takšna tehnološka režitev že obstaja. Izkazalo se je, da je na internetu dostopnih ogromno aplikacij ter rešitev, namenjenih prepoznavi teksta ter klasifikaciji fotografij, kot so TesseractOCR [1] in Google Vision [2]. Vendar nobena od njih ni specializirana za prepoznavo štartnih številk. Še najsorodnejše rešitve so tiste, ki služijo za programsko branje registerskih tablic avtomobilov, npr.

knjižnica OpenALPR [3]. Vendar ko smo jih poskusili skonfigurirati in uporabiti na našem primeru, se je izkazalo da je naš problem kompleksnejši in preprosto ne pridemo do uporabnih rezultatov.

Tako smo začeli razvijati svoj postopek. Ugotovili smo, da je najbolj optimalna strategija prepoznava štartne številke, ki jo tekač nosi na sebi. V kolikor ta problem nekoliko razdrobimo, lahko definiramo 3 glavne faze postopka: detekcija teles tekačev, izolacija regije v kateri se nahaja štartna številka ter optična prepoznava števk. Najtežji problem je bil detekcija posamezne regije, v kateri se nahaja štartna številka. Preizkusili smo različne detektorje in metode, za najboljšo rešitev, pa se je izkazal detektor posameznih delov telesa, ki za detekcijo uporablja konvolucijske nevronske mreže, ki so trenutno zelo aktualna metoda na področju računalništva in umetne inteligence. Vse faze so v diplomski nalogi podrobno opisane kot tudi nekaj dodatnih orodij, ki smo jih razvili za boljšo organizacijo fotografij ter pregled rezultatov.

V prvem poglavju smo si ogledali orodja ter postopke, ki smo jih uporabili. Vsakega od njih smo podrobno opisali, prikazali uporabo ter opisali dobljene rezultate. Predstavili smo različne tehnike binarizacije, optično prepoznavo znakov z orodjem TesseractOCR, detekcijo objektov z različnimi detektorji ter algoritem selektivno iskanje za segmentacijo tarčne regije.
V naslednjem poglavju smo se skoncentrirali na uporabo predstavljenih orodj za prepoznavo štartne številke. S shemo smo ilustrirali kako poteka celoten postopek po korakih. Najprej opišemo uporabo detektorjev objektov oz. postopke za izolacijo tarčne regije, nato pa izvedbo selektivnega iskanja za izolacijo posameznih števk. V naslednjem koraku predstavimo na kakšen način smo izločili večino nepravilnih rezultatov ter kako smo posamezne števke združili v štartno številko. Prikazali smo tudi uporabo postopka v ukazni vrstici.
V poglavju z rezultati smo primerjali uspešnost optične prepoznave znakov, v kolikor smo uporabili dodatne optimizacije, primerjali smo različne detektorje z namenom izolacije tarčne regije ter prikazali splošne rezultate detekcije

startnih številk na množici fotografij.

Poglavje 2

Uporabljene metode za obdelavo fotografij ter orodja

V celotnem postopku smo uporabili veliko orodij ter metod, ki jih je bilo najprej potrebno razumeti in nato implementirati na način, da so bila uporabna za naš primer. Ker so nekatere precej kompleksna, bo njihova uporaba in namen opisana tukaj, kasneje v opisu postopka za detekcijo, pa bodo zgolj navedena.

2.1 Binarizacija

Binarizacija [4, 5] je osnoven postopek segmentacije slike. Vsak piksel na sliki spremenimo v črno barvo, v kolikor je podan pogoj zadoščen, sicer pa v belo. Ponavadi ga uporabimo, kadar želimo na sliki ločiti posamezne objekte. Za vsako sliko moramo prilagoditi pogoj, da bomo dobili željen rezultat. V ta namen obstaja več različnih binarizacije. V nadaljevanju bom podrobneje obrazložil tiste, ki so najbolj primerne za uporabo v našem postopku.

2.1.1 Enostavna binarizacija

Najenostavnejša oblika binarizacije je enostavna pragovna binarizacija. (angl. *Binary threshold* [5]). Vsak piksel spremenimo glede na dan pogoj. Oznaka

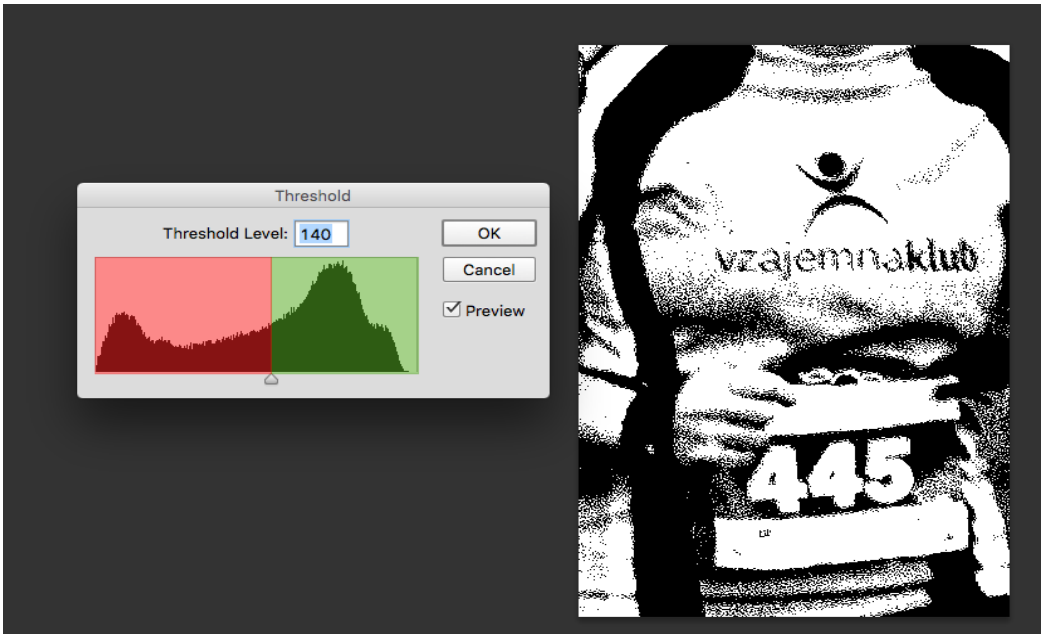
src(x, y) označuje piksel na poziciji x,y na izvorni fotografiji, *dst(x,y)* pa piksel na binarizirani fotografiji, ki se nahaja na poziciji *x, y*.

$$dst(x,y) = \begin{cases} 1, & \text{če } src(x,y) > prag \\ 0, & \text{sicer} \end{cases} \tag{2.1}$$

V kolikor je vrednost piksla na izvorni sliki na poziciji x,y večja od določene meje, mu priredimo vrednost 1, sicer pa 0.

Problem te preproste metode je, da v kolikor hočemo ločiti posamezne objekte na poljubni fotografiji, moramo prilagajati mejo za vsako fotografijo posebaj, zato metoda za našo uporabo ne pride v poštev.

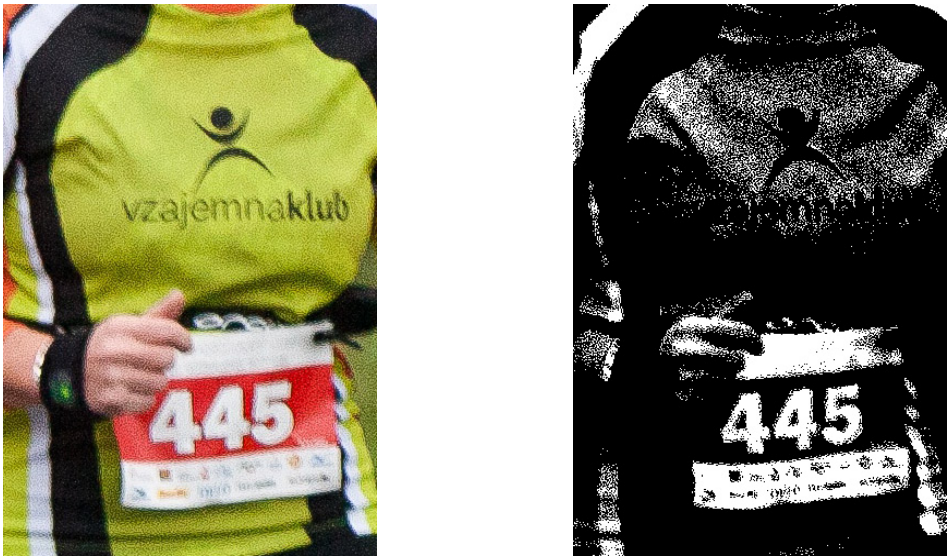
Na spodnji fotografiji (slika 2.1) je prikazan dialog za konfiguracijo binarizacije v programu Adobe Photoshop CC. Vrednost meje je nastavljena na 140. Piksli, ki imajo večjo vrednost od 140 (na histogramu označeni z zeleno barvo) bodo postali črni, tisti z nižjo vrednostjo pa beli (na histogramu označeni z rdečo barvo).



Slika 2.1: Primer binarizacije v programu Adobe Photoshop CC.

Na slikah 2.2 in 2.3 lahko vidimo delno uspešne binarizacije slike, z upo-

rabo metode enostavne binarizacije.



Slika 2.2: Primer delno uspešne binarizacije. Na levi sliki je izrez iz izvorne fotografije, na desni pa binarizirana verzija izreza, z mejno vrednostjo 200.

Slika 2.3: Primer delno uspešne binarizacije. Na levi sliki je izrez iz izvorne fotografije, na desni pa binarizirana verzija izreza, z mejno vrednostjo 170.

2.1.2 Binarizacija Otsu

Otsu metoda binarizacije [5] je poimenovana po izumitelju, japonskem matematiku Nobuyuki Otsu. V primerjavi z binarno metodo ima veliko prednost, saj ne potrebuje nobenega ročno definiranega parametra. Temelji na predpostavki, da lahko piksle na sliki razdelimo v 2 skupini glede na intenziteto in na podlagi tega izračuna optimalno mejno vrednost praga.

V veliki večini primerov privede do optimalnih rezultatov (slike 2.4 in 2.5), zato je najprimernejša metoda za našo uporabo.

Slika 2.4: Primer uspešne binarizacije. Na levi sliki je izrez iz izvorne fotografije, na desni pa binarizirana verzija izreza z metodo Otsu.

Slika 2.5: Primer uspešne binarizacije. Na levi sliki je izrez iz izvorne fotografije, na desni pa binarizirana verzija izreza z metodo Otsu.

2.1.3 Primerjava metod

V primerih, ko pikslov ne moremo razdeliti v 2 intenzitetni skupini, metoda Otsu ne privede do optimalnih rezultatov. Na slikah 2.6 in 2.7 lahko vidimo, da bi izračunan prag binarizacije moral biti precej višji. Ker pa na to ne moremo vplivati, se bomo zadovoljili z rezultatom Otsujeve metode ter kasneje v postopku izvedli postopek erozije (opisan v sekciji 2.2), ki rezultatov v nobenem primeru ne poslabša, temveč jih v določenih primerih izboljša.

Slika 2.6: Primerjava izvorne slike, Otsu binarizacije ter Otsu binarizacije s 7 iteracijami erozije.

Slika 2.7: Primerjava izvorne slike, Otsu binarizacije ter Otsu binarizacije s 4 iteracijami erozije.

2.2 Optična prepoznava znakov

Za optično prepoznavo znakov (angl. optical character recognition) smo uporabili programsko opremo TesseractOCR [1]. Je brezplačna za uporabo ter na voljo vsem najbolj priljubljenim operacijskim sistemom. Velja za enega najbolj natančnih prosto dostopnih sistemov za OCR, kljub temu da začetki projekta segajo v leto 1985 k podjetju Hawlett Packard. Po desetih letih razvoja, so programerji večino kode prepisali iz jezika C v C++ z namenom podpore operacijskemu sistemu Windows. Naslednji pomemben mejnik v razvoju orodja je leto 2005, ko je Hawlett Packard spremenil projekt v odprtokodnega. Od leta 2006 naprej za razvoj in vzdrževanje skrbi Google pod nadzorom prvotnega avtorja Raya Smitha.

Uporaba in stikala

TesseractOCR nima grafičnega uporabniškega vmesnika. Uporaba je možna samo iz ukazne vrstice. Ker je takšna uporaba za nas povsem neprimerna, smo napisali aplikacijski programski vmesnik (angl. API). S pomočjo programskega jezika Python uporabimo metode OCR v višjenivojskem jeziku in pridobivamo njihove rezultate. Zaradi neotimiziranega postopka smo s tem tudi povečali čas posamezne detekcije. Za detekcijo posamezne številke v povprečju porabi 0.317 sekunde (izračunano na podlagi 30 detekcij).

Osnovna sintaksa ukaza je sledeča:

```
tesseract imagename outputbase [-l lang][--psm pagesegmode]
    [configfiles...]
```

Primer uporabe:

```
tesseract tmp/TYH4806B0X.jpg tmp/tesseract_outCW4RJ8YP50
    -psm 10 -c tessedit_char_whitelist=0123456789
    -c tessedit_create_hocr=1
```

Za detekcijo posameznih številk uporabimo stikalo *-psm* z vrednostjo 10. Izbira vrednsti 10 pripravi sistem za prepoznavo enega tekstovnega znaka.

S pomočjo stikal *-c tessedit_char_whitelist* definiramo veljavne znake za zaznavo. V našem primeru so le numerični znaki. Zadnje stikalo *-c tessedit_create_hocr=1* pa zagotovi, da bo vrnjeni rezultat v formatu HOCR.

Format HOCR [6, 7] je splošen odprt standard za reprezentacijo teksta, ki je bil prepoznan po metodi OCR. Primeren je za nadaljnjo obdelavo in ekstrakcijo željenih podatkov. V spodnjem primeru rezultata lahko na 21. vrstici izluščimo verjetnost pravilne zaznave (angl. *confidence*), ki se nahaja pod atributom *x_wconf*. Sam rezultat pa se nahaja v vrstici 22. V našem primeru lahko mirno odstranimo značke <*span*> ter <*em*> in tako dobimo za rezultat številko 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title></title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  <meta name='ocr-system' content='tesseract 3.04.01' />
  <meta name='ocr-capabilities'
        content='ocr_page ocr_carea ocr_par ocr_line ocrx_word'/>
</head>
<body>
  <div class='ocr_page' id='page_1' title='image "tmp/PBYAGL1TVW.jpg";
    bbox 0 0 20 25; ppageno 0'>
   <div class='ocr_carea' id='block_1_1' title="bbox 8 12 16 22">
    <p class='ocr_par' dir='ltr' id='par_1_1' title="bbox 8 12 16 22">
     <span class='ocr_line' id='line_1_1'
        title="bbox 8 12 16 22; baseline -0 0;
            x_size 20; x_descenders 5; x_ascenders 5">
        <span class='ocrx_word' id='word_1_1'
        title='bbox 8 12 16 22; x_wconf 66' lang='eng'>
        <strong><em>4</em></strong>
        </span>
     </span>
    </p>
   </div>
  </div>
 </body>
</html>
```

Slika 2.8: Primer rezultata optične prepoznave znakov, v formatu HOCR.

Ker pa je zgornji postopek precej *neokreten* za višjenivojsko uporabo, smo v programskem jeziku Python napisali preprost aplikacijski programski vmesnik (angl. API), ki poenostavi uporabo orodja (slika 2.9).

```
class TesseractAPI(object):

    @classmethod
    def recognise_single_char(cls, path_or_cv2_image):

        if type(path_or_cv2_image) != np.ndarray:
            cv2_image = cv2.imread(path_or_cv2_image)
        else:
            cv2_image = path_or_cv2_image

        return cls.recognise(cv2_image, 10)

    @classmethod
    def recognise(cls, path_or_cv2_image, psm_mode):
        # v kolikor je slika podana kot argument,
        # jo shrani na začasno lokacijo,
        # saj moramo orodju podati pot do datoteke.
        # Sestavi tesseract ukaz (prikazano zgoraj)
        # Ujemi vrnjeno vrednost (v formatu HOCR)
        # in izlušči dobljen rezultat
        # Izbriši ustvarjene datoteke
        # ...
    # ...
```

Slika 2.9: Del implementacije programskega vmesnika za lažjo uporabo orodja TesseractOCR v jeziku Python.

Tako detekcijo, na podani fotografiji, izvedemo s preprostim klicem:

```
number, prob = TesseractAPI\
    .recognise_single_char(digit_candidate_cv2_image)
```

Slika 2.10: Klic visokonivojske funkcije, ki vrne rezultat optične zaznave.

Zaznava posameznih števk

TesseractOCR v večini primerov vrne zelo dobre rezultate, včasih pa precej zgreši, kljub temu da vrne visoko verjetnost zaznanega rezultata. To lahko vidimo v prvi vrstici tabele 2.11. V ta namen lahko implementiramo nekaj izboljšav, ki povečajo verjetnost pravilnega rezultata, vendar s tem povečamo čas prepoznave. Implementirane izboljšave so erozija in rotacija posameznega znaka, ki so podrobno predstavljene v nadaljevanju.

Slika	Zaznana Števka	Verjetnost
	1	68 %
	4	93 %
	4	88 %
	9	92 %

Slika 2.11: Tabela neizboljšanih rezultatov.

V tabeli lahko vidimo, da prva vrstica vsebuje povsem napačen rezultat, čeprav je verjetnost pravilne zaznave precej visoka. To so anomalije, ki nastanejo pri postopku binarizacije. V nadaljevanju bomo opisali dva postopka izboljšav, ki omenjeno anomalijo omilita (sekcija 2.2.1).

Izboljšava z rotacijo

Orodje je precej močno občutljivo na rotacijo znaka. Kot je prikazano v spodnjih tabelah 2.12 in 2.13, lahko z rotacijo slike znaka za nekaj stopinj spremenimo pravilnost in verjetnost pravilnega rezultata iz 0% na 90%.

Zato za vsako sliko ki jo želimo zaznati, pripravimo več verzij, vsako rotiramo za določen kot, kot je prikazano v tabelah 2.12 in 2.13. Za to poskrbimo že na nivoju aplikacije in tam tudi sprocesiramo rezultate in poiščemo najverjetnejšega.

Slika	Rotacija	Zaznana Števka	Verjetnost
	-20 °	4	90 %
	-10 °	4	86 %
	0 °	4	87 %
	10 °	/	/
	20 °	/	/

Slika 2.12: Tabela rezultatov z uporabo izboljšave z rotacijo.

Omenjena izboljšava je zelo pomembna, saj so številke tekačev le v redkih primerih v povsem ravnem položaju.

Slika	Rotacija	Zaznana Števka	Verjetnost
	-20 °	6	68 %
	-10 °	5	79 %
	0 °	5	89 %
	10 °	5	94 %
	20 °	5	85 %

Slika 2.13: Tabela rezultatov z uporabo izboljšave z rotacijo.

Erozija

Pred izvedbo same prepoznave moramo sliko binarizirati. To lahko namesto nas naredi tudi TesseractOCR, vendar v dokumentaciji svetujejo, da to

storimo sami. Pri sami binarizaciji pa velikokrat ostanejo na sliki še drugi manjši deli, ki niso del znaka in lahko zmanjšajo verjetnost pravilnega rezultata. Zato uporabimo morfološko transformacijo imenovano erozija (angl. *erude*), ki je implementirana v standardni verziji knjižnice *OpenCV (cv2)*[8].

```
cv2.erode(img, kernel, iterations = 1)
```

Metodi podamo tri argumente. Prvi argument določa sliko, drugi velikost okna, tretji pa število iteracij algoritma - več iteracij, več erozije črnih pikslov.

Podobno kot pri izboljšavi z rotacijo, tudi v tem primeru naredimo več prepoznav, kot prikazujejo tabele 2.15 in 2.16. Kot končni rezultat izberemo tisto prepoznavo, ki vrne največjo verjetnost pravilnega rezultata.

```
import cv2
import numpy as np

img = cv2.imread('foobar.png', 0)
kernel = np.ones((5, 5), np.uint8)
eroded = cv2.erode(img, kernel, iterations = 1)
```

Slika 2.14: Implementacija postopka erozije s pomočjo knjižnice OpenCV.

Slika	Iteracije	Zaznana števka	Verjetnost
	0	/	/
	1	5	88 %
	2	5	84 %
	3	5	61 %
	4	/	/

Slika 2.15: Izboljšava prepoznave števk s postopkom erozije.

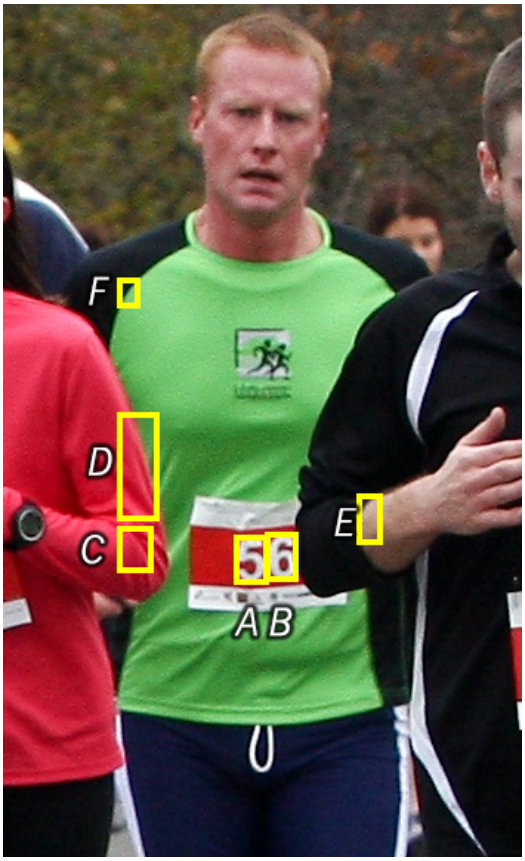
Slika	Iteracije	Zaznana števka	Verjetnost
	1	5	59
	2	5	57 %
	3	5	85 %
	4	5	87 %
	5	5	76 %
	6	5	54 %
	7	/	/

Slika 2.16: Izboljšava prepoznave števk s postopkom erozije.

2.2.1 Filtriranje rezultatov detekcije glede na hevristična pravila

Na fotografiji 2.17 smo izvedli prepoznavo znakov. Metoda je odkrila veliko regij (rumeni pravokotniki) z veliko verjetnostjo pravilno zaznave števke. Označili smo jih s črkami od A do F. Ampak rezultat je velikokrat le posledica binarizacije, ki smo jo izvedli nad celotno regijo, v kateri pričakujemo da se nahaja štartna številka.

Z uporabo metod ki jih bomo predstavili v nadaljevanju, lahko večino omenjenih anomalij odstranimo. Vse metode bodo predstavljene na fotografiji slike 2.17.



Slika 2.17: Fotografija z zaznanimi rezultati.

Števke z visoko verjetnostjo

Prvi in najvažnejši korak, ki je tudi potencialna šibka točka celotnega postopka, je da določimo za katere rezultate smo prepričani, da so pravilni. Poimenovali jih bomo *števke z visoko verjetnostjo pravilne detekcije* (VVPD). Na množici selekcioniranih števk, bomo izračunali določene lastnosti, ki jih bomo nato primerjali z vsakim rezultatom (regije označene z črkami). V kolikor izračun preveč odstopa, bomo rezultat označili za neveljaven. Prag minimalne verjetnosti lahko poljubno izbiramo. V našem primeru, smo na podlagi preizkušanja, vrednost nastavili na $p_{min} = 80\%$.

števka	Verjetnost	širina □	višina □	X	Y
5 (A)	87 %	20px	30px	2628	885
6 (B)	74 %	19px	30px	2649	883
1 (E)	74 %	14px	31px	2711	857
4 (F)	73 %	12px	17px	2549	711
3 (D)	64 %	25px	70px	2549	802
/ (C)	/	/	/	/	/

Tabela 2.1: Podatki rezultatov, ki so bili prepoznani kot števke na sliki 2.17.

Razlika v višini

Vse števke ki so na sliki, imajo približno enako višino (tabela 2.1). Tudi v primeru, da je slika poševna, bodo višine še vedno približno enake, saj se nahajajo v isti ravnini in so pri vseh tekačih enako velike. Števke z visoko verjetnostjo pravilnega rezultata si vzamemo *za zgled* in vse rezultate primerjamo z njihovo povprečno višino, pri čemer dovolimo določen odstotek odstopanja.

Poljubno določimo prag visoke verjetnosti pravilne detekcije števke ter prag maksimalne dovoljene razlike od povprečne velikosti števk z visoko verjetnostjo pravilne detekcije

$$p_{min} = 80\% \qquad \Delta\overline{v}_{max} = 10\%$$

Izračunamo povprečje višin pravokotnikov števk, ki so bile zaznane z visoko verjetnostjo pravilne detekcije.

$$\overline{v} = \sum_{x_p > p_{min}} x_v$$

Nato za vsako števko izračunamo razliko od poprečne velikosti števk z visoko verjetnostjo pravilne detekcije.

$$\Delta v = \overline{v} - x_v$$

$$\Delta v = (\overline{v} - x_v) * 100/\overline{v}$$

V kolikor je $\Delta v > \Delta\overline{v}_{max}$ potem to številko označimo kot lažno pozitivni rezultat (angl. *false positive*).

številka	Verjetnost	širina □	višina □	X	Y	Δv	Δv
5 (A)	87 %	20px	30px	2628	885	0px	0 %
6 (B)	74 %	19px	30px	2649	883	0px	0 %
1 (E)	74 %	14px	31px	2711	857	1px	3,3 %
4 (F)	73 %	12px	17px	2549	711	13px	43 %
3 (D)	64 %	25px	70px	2549	802	40px	133 %
/ (C)	/	/	/	/	/	/	/

Tabela 2.2: Tabela filtriranih številk glede na razliko v višini.

Glede na podatke v tabeli 2.2 je izračunana povprečna višina 30px (višina rezultata A, ki edini zadošča pogoju, da je verjetnost pravilnega rezultata večja od 80%). Vrstice označene z rdečo barvo presegajo dovoljeno 10% razliko v višini (zadnji stolpec v tabeli) in so zato označene kot lažno pozitivni rezultat.

Horizontalna oddaljenost številk

V tem koraku, bomo izločile detektirane na podlagi horizontalne oddaljenost od številk z visoko verjetnostjo pravilne detekcije. Vse številke na sliki se nahajajo v gruči. Torej lahko izločimo tiste rezultate, ki imajo visok odstotek pravilnosti zaznanega rezultata, hkrati pa veliko oddaljenost od gruče rezultatov z visoko verjetnostjo pravilne detekcije.

S preizkušanjem, določimo prag visoke verjetnosti pravilne detekcije številke ter faktor dovoljene napake (m).

$$p_{min} = 80\% \qquad m = \pm 30\% = 1,3$$

Izračunamo povprečje širin pravokotnikov števk, ki so bile zaznane z visoko verjetnostjo pravilne detekcije.

$$\overline{w} = \sum_{x_p > p_{min}} x_w$$

$$\overline{w}_{max} = \overline{w} * m$$

Za vsako števko izračunamo minimalno oddaljenost od števk z visoko verjetnostjo pravilne detekcije. (V tabeli 3.3, stolpec $\|x_{min}\|$) To storimo z uporabo funkcije:

```
def get_minimal_high_prob_dist(digit_list):
    high_prob_digits = [
            x for x in digit_list
            if x['prob'] > P_MIN
    ]

    for digit in digit_list:

            # highest Integer available in python!
        min_dist = sys.maxint

        for high_prob_digit in high_prob_digits:

            x_diff = abs(high_prob_digit['x'] - digit['x'])
            if x_diff <= min_dist:
                min_dist = x_diff

        digit['high_prob_min_x_dist'] = min_dist

    return digit_list
```

Nato primerjamo minimalno razdaljo do števke z visoko verjetnostjo pravilne detekcije ter povprečje širin števk z visoko verjetnostjo pravilne detek-

cije. V enačbo vključimo tudi poljuben faktor napake, ki jo dovoljujemo pri izračunu.

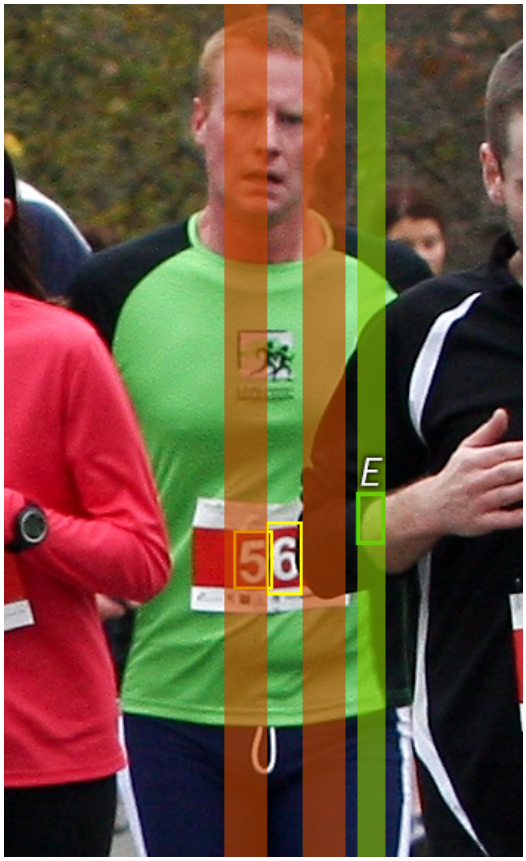
$$\Delta d = \|x_{min}\| - \overline{w}$$

številka	i	Verjetnost	širina □	X	$\|x_{min}\|$	$\|x_{min}\| < \overline{w}_{max}$
5 (A)	0	87 %	20px	2628	0 px	DA
6 (B)	1	74 %	19px	2649	21 px	DA
1 (E)	2	74 %	14px	2711	83 px	NE
4 (F)	3	73 %	12px	2549	79 px	NE
3 (D)	4	64 %	25px	2549	79 px	NE
/ (C)	5	/	/	/	/	/

Tabela 2.3: Tabela filtriranih števk glede na horizontalno ddaljenost od števk VVPD.

Glede na podatke v tabeli 2.3 je izračunana povprečna širina pravokotnikov 20px (širina rezultata A, ki edini zadošča pogoju, da je verjetnost pravilnega rezultata večja od 80 %). V primeru E je minimalna oddaljenost 83px, kar presega maksimalno dovoljeno oddaljenost 26px. Zato je rezultat E označen kot lažno pozitivni rezultat. Po enakem principu izločimo rezultate F in D.

Zgornje rezultate lahko tudi vizualiziramo na fotografiji. Zeleni pas označuje x koordinate kandidata, za katerega sistem trdi, da z 74% verjetnostjo ustreza številki 1 (na fotografiji označena s črko E). Rdeča pasova nakazujeta kje bi se moral kandidat E nahajati, da bi bila njegova horizontalna razlika do najbližje števke z visoko verjetnostjo pravilne detekcije zadostna. (Vizualizacija ustreza iteraciji izračunu razdalj označenim s črko B, kjer uporabljamo številko 6 kot števko z visoko verjetnostjo pravilne detekcije.)



Slika 2.18: Vizualizacija filtriranja na podlagi horizontalne oddaljenosti.

Vertikalna oddaljenost števk

Uporabimo enak princip, enačbe ter funkcije kot pri horizontalni oddaljenosti samo da uporabljamo y koordinate točk ter višino namesto širine. Zaradi velike podobnosti s prejšnjim načinom, bomo opisne podrobnosti izpustili in navedli samo enačbe.

$$p_{min} = 80\% \qquad m = \pm 10\% = 1,3$$

$$\overline{h} = \sum_{x_p > p_{min}} x_h$$

$$\overline{h}_{max} = \overline{h} * m$$

številka	Verjetnost	višina ▭	Y	$\|y_{min}\|$	$\|y_{min}\|<\overline{h}_{max}$
5 (A)	87 %	30px	885	0px	DA
6 (B)	74 %	30px	883	2px	DA
1 (E)	74 %	31px	857	28px	DA
4 (F)	73 %	17px	711	174px	NE
3 (D)	64 %	70px	802	83px	NE
/ (C)	/	/	/	/	/

Tabela 2.4: Tabela filtriranih števk glede na vertikalno oddaljenost od števk z VVPD.

Glede na podatke v tabeli je izračunana povprečna višina pravokotnikov 30px (širina rezultata A, ki edini zadošča pogoju, da je verjetnost pravilnega rezultata večja od 80 %). V primeru F, je minimalna oddaljenost 174px, kar presega maksimalno dovoljeno oddaljenost 39px. Zato je rezultat F označen kot lažno pozitivni rezultat. Po enakem principu izločimo tudi rezultat D.

Rumeni pas označuje y koordinate kandidata, za katerega sistem trdi, da z 73-% verjetnostjo ustreza številki 4 (na fotografiji označena s črko F). Rdeča pasova nakazujeta kje bi se moral kandidat F nahajati, da bi bila njegova horizontalna razlika do najbližje števke z visoko verjetnostjo pravilne detekcije zadostna. (Vizualizacija ustreza iteraciji izračunu razdalj označenim z črko B, kjer uporabljamo številko 6 kot števko z visoko verjetnostjo pravilne detekcije.)

Slika 2.19: Vizualizacija filtriranja vertikalne oddaljenosti.

Rezultati z uvedbo dodatnih hevrističnih pravil

Če izvedemo celoten postopek detekcije in primerjamo rezultate s tistimi, ko izvedemo izboljšave z uporabo rotacije in erozije oz. jih izpustimo, pridemo do velikih razlik v natančnosti prepoznave. (Prikazano v sekciji 4.1, tabela 4.2). Poudariti je potrebno, da se rezultati zelo razlikujejo med posameznimi fotografijami. Omenjene izboljšave pridejo najbolj do izraza pri fotografijah, ki jih je zajel fotograf, ki se je vozil na motorju, saj v tem primeru večina fotografij ni povsem pravilno zasukanih.

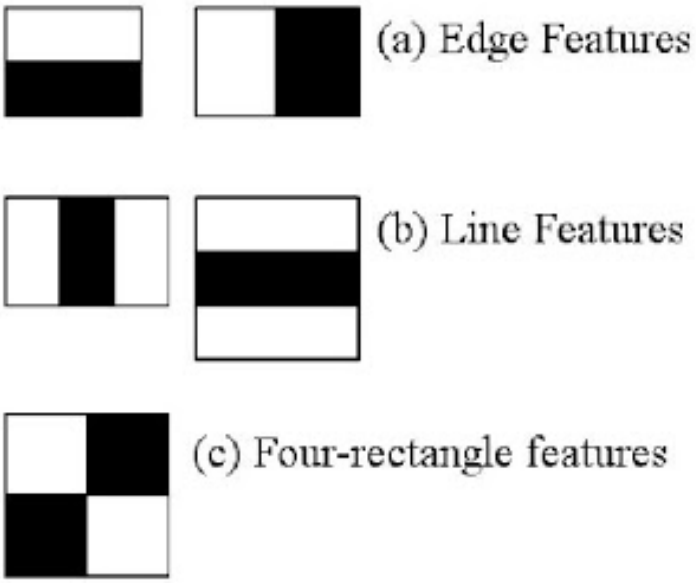
2.3 Detekcija objektov

2.3.1 HAAR detektorji

HAAR detektor [9] je detektor objektov na sliki, ki za detekcijo uporablja klasifikatorje v katerih je ustrezen opis takoimenovanih HAAR značilk (*angl. Haar-like features*) na podlagi katerih lahko prepoznamo določen objekt.

Metodo sta razvila Paul Viola in Michael Jones v članku *Rapid Object Detection using a Boosted Cascade of Simple Features*[10] v letu 2001 in je v računalniških krogih poznana pod imenom *Viola-Jones detektor*. Postopek uporablja metode strojnega učenja, s pomočjo katerih na podlagi velike množice pozitivnih slik (tiste slike ki vsebujejo tarčni objekt) ter negativnih slik (tiste slike ki ne vsebujejo tarčnega objekta) izračuna klasifikator, ki ga kasneje lahko uporabimo za hitro in efektivno detekcijo objekta na sliki. Algoritem iz vsake podane fotografije izračuna HAAR značilk z uporabo mask (slika 2.20). Najbolj uporabljene so

- Maska za detekcijo robnih lastnosti
- Maska za detekcijo linijskih lastnosti
- Maska za detekcijo pravokotnih lastnosti



Slika 2.20: HAAR maske [9].

Vsako izmed njih si lahko predstavljamo kot *okno*, ki drsi čez sliko, ter izračuna vrednost dane funkcije za piksle, ki jih zaobjema beli del pravokotnika, ter vrednost dane funkcije za piksle, ki zaobjema črni del pravokotnika.

S kompleksnim filtriranjem pridobljenih izračunov definiramo klasifikator, ki vsebuje opis lastnosti. Kadar izvajamo detekcijo objekta preizkušamo vsako značilko posebej. Najprej preizkusimo lastnosti iz prve faze. V kolikor ena od njih za sliko ne velja, zaključimo s prepoznavo, saj slika ne vsebuje iskanega objekta. Iz tega sledi, da slika vsebuje iskan objekt v kolikor vsebuje vse značilke, ki jih definira klasifikator.

Knjižnica OpenCV ima implementirano funkcijo za uporabo omenjenega klasifikatorja. Uporaba je prikazana na sliki 2.21

```
import cv2

cascade_file_path = "opencv/data/haarcascades/haarcascade_fullbody.xml"
image_path = "foo.jpg"

scale_factor = 1.1
min_neighbours = 1

# Sliko odpremo z ogrodjem OpenCV
cv2_img = cv2.imread(image_path)

# detektor zahteva sivinsko sliko (ne RGB!)
cv2_img_bw = cv2.cvtColor(cv2_img, cv2.COLOR_BGR2GRAY)

face_cascade = cv2.CascadeClassifier(cascade_file_path)
results = face_cascade.detectMultiScale(
        cv2_img_bw, scale_factor, min_neighbours)
```

Slika 2.21: Primer uporabe OpenCV HAAR detektorja.

Kreiranja in učenja lastnih modelov nismo potrebovali, ker je na spletu na voljo veliko enostavno dosegljivih modelov, učenih na velikih podatkovnik zbirkah, ki dajejo zelo dobre rezultate.

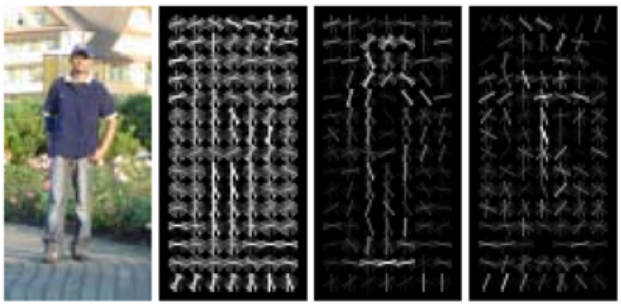
2.3.2 HOG detektorji

Eden bolj priljubljenih detektorjev oblik je HOG detektor [11] (angl. *Histogram of Oriented Gradients detector*). Podobno kot HAAR detektor, iz tarčnih slik izračuna določene značilnosti, ki dano sliko generalizirajo. Te značilnosti tvorijo globalen objekt, ki ga imenujemo HOG deskriptor (slika 2.22).

Za detekcijo objektov, uporabimo drseče okno. Za vsako regijo fotografije izračunamo HOG deskriptor, ki ga nato podamo klasifikatorju, ki izračuna ali podan deskriptor ustreza določenemu objektu ali ne.

Postopek je bil teoretično razvit že v letu 1986, ampak je v masovno upo-

rabo vstopil šele leta 2005. Takrat sta ga Navneet Dalal in Bill Triggs [11] predstavila na največji svetovni konferenci za računalniški vid in prepoznavo vzorcev iz slik (angl. *Conference on Computer Vision and Pattern Recognition*). Ostredotočila sta se na detekcijo človeških teles iz fotografij. Nato sta svoj članek razširila na detekcijo teles, vozil in živali v video posnetku [12].



Slika 2.22: Primer HOG deskriptorja, prikazan v izvornem članku [11].

2.3.3 Detekcija pozicije telesa z uporabo nevronskih mrež

Še do pred kratkim je bila najbolj priljubljena metoda za detekcijo objektov na fotografijah HAAR detektor (opisan v pod-sekciji 2.3.1. V zadnjih dveh letih, pa se je zelo močno razširilo področje uporabe konvolucijskih nevronskih mrež (angl. *Convolutional Neural Network - CNN*) za klasifikacijo ter detekcijo objektov na fotografijah, vendar je v primerjavi s HAAR detektorji uporaba nekoliko kompleksnejša.

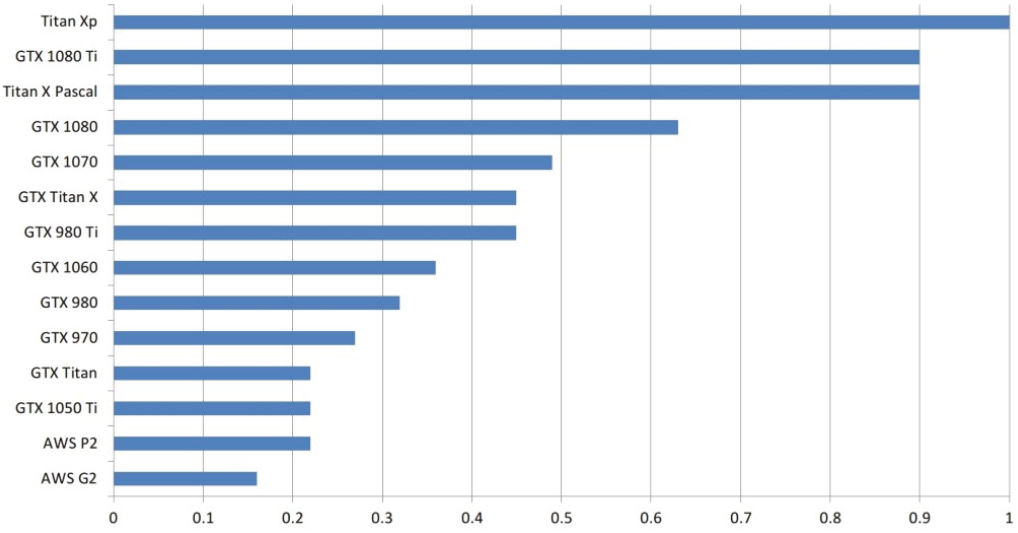
Strojna oprema

Tako za učenje in definicijo modelov kot samo uporabo programske opreme, potrebujemo zelo zmogljivo strojno opremo, ki za izračune uporabljajo grafične kartice, saj na tak način močno dvignemo nivo pararelizacije in s tem močno zmanjšamo skupen čas izračunov. Primer klučnih komponent ter cen visokozmogljivega računalnika, primernega za kreiranje modelov ter izvedbo detekcije objektov vidimo v tabeli 2.23.

Matična plošča	Asus X99-E WS workstation	600€
Procesor	Intel i7 5930K 6 Core 3.5 GHz LGA	490€
RAM pomnilnik	64GB DDR4 Kingston 2133Mhz (8x8GB)	630€
Grafična kartica	3 x NVIDIA GTX TITAN-X 12GB	1700€
Napajalnik	Corsair AX1500i (1500Watt) 80 Plus Titanium	410€

Slika 2.23: Primer strojne konfiguracije računalnika, primernega za uporabo konvolucijskih nevronskih mrež.

Ključna strojna komponenta je grafična kartica (angl. *graphics processing unit - GPU*), saj le ta omogoča visoko stopnjo pararelizacije in s tem krajše čase izvedbe danih operacij. Na trgu je velika izbira kartic, prevladujejo pa modeli proizvajalca Nvidia. To je predvsem zato, ker ima veliko programske opreme na voljo podporo le za kartice proizajalca Nvidia [13]. Primerjava zmogljivosti najbolj priljubljenih grafičnih kartic se nahaja na sliki 2.24.



Slika 2.24: Primerjava zmogljivosti nekateri bolj priljubljenih grafičnih kartic. [14]

Zaradi visokih cen nakupa opreme je enostavnejše najeti primeren strežnik,

ki ga konfiguriramo po svojih potrebah nato pa plačujemo glede na ure procesiranja. Na tak način lahko za precej nižjo ceno uporabljamo vrhunsko strojno opremo, kadar jo potrebujemo. Taki ponudniki GPU gostovanj so npr. Redstation [15], OVH [16], Hetzner [17]...

Programska oprema in okolje

Postopek smo razvili na Applovem operacijskem sistemu OS X, vendar mislim da bi bilo vse skupaj precej lažje, v kolikor bi za okolje izbral katero izmed popularjejših distribucij Linuxa, kot je Ubuntu. S tem bi se izognil določenim problemom pri namestitvi okolja ki je precej kompleksno.

Za uporabo HAAR detektorja načeloma zadostuje operacijski sistem ter povsem standardna namestitev programskega jezika Python. Dodamo še knjižnice kot so PIL (*Python image library*) ter CV2 (*OpenCV 2*). Vse so enostavno dosegljive preko urejevalnika paketov *pip*. V kolikor sledimo navodilom, ki jih enostavno najdemo na internetu, lahko z namestitvijo programske opreme opravimo zelo hitro. Tako lahko čas posvetimo vsebinskim problemom ter programiranju detekcije, namesto da se ukvarjamo s samim okoljem.

Okolje za uporabo konvolucijskin nevronskih mrež z namenom detekcije objektov, pa je precej kompleksnejše (z uporabo knjižnice Caffe in GPU podpore). Gradnikov je precej več, hkrati pa je vsak izmed njih tudi neprimerljivo kompleksnejši, saj ne govorimo zgolj o knjižnici ali modulu, temveč o celotnem programskem paketu, ki vsebuje tako nizkonivojsko kot visokonivojsko kodo:

- **Operacijski sistem** - priporočena je uporaba operacijskega sistema Linux, saj je večina dokumentacij različnih knjižnic ter odrodij, privzema uporabo le tega. S nekaj sreče lahko uprabljamo tudi Applov OS X, saj je grajen na isti osnovi kot Linux. Uporaba sistema Microsoft Windows je manj priljubljena, ker marsikatera knjižnica oz. paket tega operacijska sistema ne podpira oz. je potrebna dodatna konfiguracija sistema [18].

- **GPU ogrodje za pararelizacijo ter podporo nevronskih mrež** - Sama grafična kartica ni kaj prida uporabna, v kolikor operacijski sistem ni sposoben dobro komunicirati z njo. Zato potrebujemo programsko opremo, ki omogoča pararelizacijo operacij z uporabo grafične kartice. Priporoča se uporaba grafičnih kartic proizvajalca NVIDIA ter uporaba ogrodja za pararelizacijo CUDA [19].

  V našem primeru bi grafično kartico uporabljali za izračune nevronskih mrež. Zato potrebujemo še dodaten paket cuDNN, ki podpira in poenostavi uporabo le teh. Pri uporabi oz. detekciji, se lahko grafičnemu procesorju povsem izognemo. Izvajanje operacij bo počasnejše in ponavadi neprimerno za produkcijsko uporabo. Pri učenju modela pa je uporaba GPUja obvezna, saj bi v nasprotnem primeru učenje posameznega modela lahko trajalo tudi več mesecev.

- **Ogrodje za globoko učenje** (*Deep Learning framework*) - Na spletu lahko najdemo veliko knjižnic za globoko učenje. Trenutno so najbolj popularne Caffe, Keras, TensorFlow, Theano in Torch. V okolju MATLAB pa je najbolj priljubljeno orodje MatConvNet. Za naš primer uporabe nevronskih mrež za detekcijo objektov, se favorizira ogrodje Caffe.

  Tako kot vsa druga našteta ogrodja, je tudi to napisano v jeziku C++, uporabljamo pa ga z uporabo aplikacijskega programskega vmesnika v programskem jeziku Python. Sama uporaba je na začetku precej težavna, saj je dokumentacija precej skopa.

- **Implementacija detekcije**

  Implementirali smo detektor [20], ki je bil predstavljen na najbolj znani svetovni konferenci za računalniški vid leta 2016 (*CVPR'16*). Koda z navodili se nahaja na Github repozitoriju z imenom *convolutional-pose-machines-release* [21].

  – **Inicializacija Caffe** Prvi korak je da fotografijo spravimo v pri-

meren format (*np.float32*) ter inicializiramo knjižnico Caffe s primernimi parametri ter definiramo katere modele bomo uporabili.

Priloženi modeli so se izkazali za zelo natančne. Naučeni so na velikih bazah fotografij. Najbolj poznane med njimi sta *MPII Human Pose Dataset*[22], ki vsebuje približno 25000 fotografij oz. 40000 oseb, ter *Frames Labeled In Cinema - FLIC*[23] ki vsebuje 5003 fotografij, pridobljenih iz filmov.

Najpomembnejši parametri, ki jih je potrebno definirati so *use_gpu* in *GPUdeviceNumber*. Skupaj definirata ali bomo uporabljali centralno procesno enoto ali grafično kartico ter katero od njih, v primeru da jih imamo več. V našem primeru smo oba nastavili na vrednost 0, saj smo med razvojem uporabljali centralno procesno enoto.

```
use_gpu = 0  # Uporaba grafične kartice?
GPUdeviceNumber = 0  # indentifikator grafične kartice
```

– **Detekcija ljudi** Najprej detektiramo posamezne osebe, s priloženim modelom za detekcijo oseb *pose_iter_70000.caffemodel*.

```
# Model za detekcijo oseb
caffemodel_person = \
        'lib/convolutional-pose-machines-release/model/'
        '_trained_person_MPI/pose_iter_70000.caffemodel'
```

Za detekcijo sliko zmanjšamo na velikost slik na podlagi katerih je bil naučen model. V tem koraku predpostavljamo, da višina oseb ne presega višine slike in hkrati ni premajhna. Inicializiramo nevronsko mrežo ter izvedemo izračun.

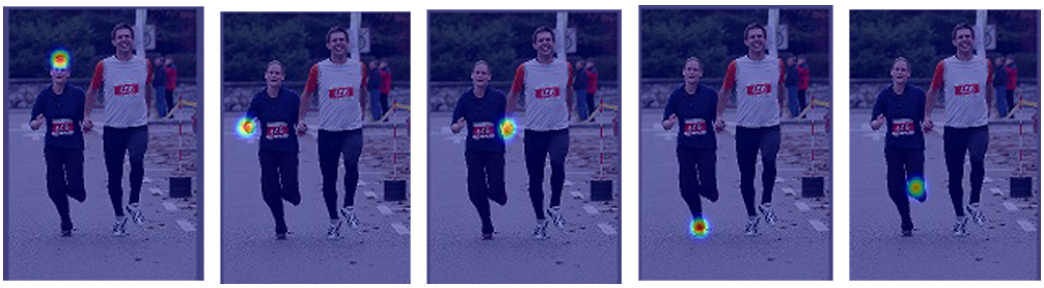
Kot rezultat dobimo točke, ki prestavljajo centre pozicij telesa.

Slika 2.25: Maksimumi verjetnosti detekcije telesa.

Slika 2.26: Maksimumi verjetnosti detekcije telesa.

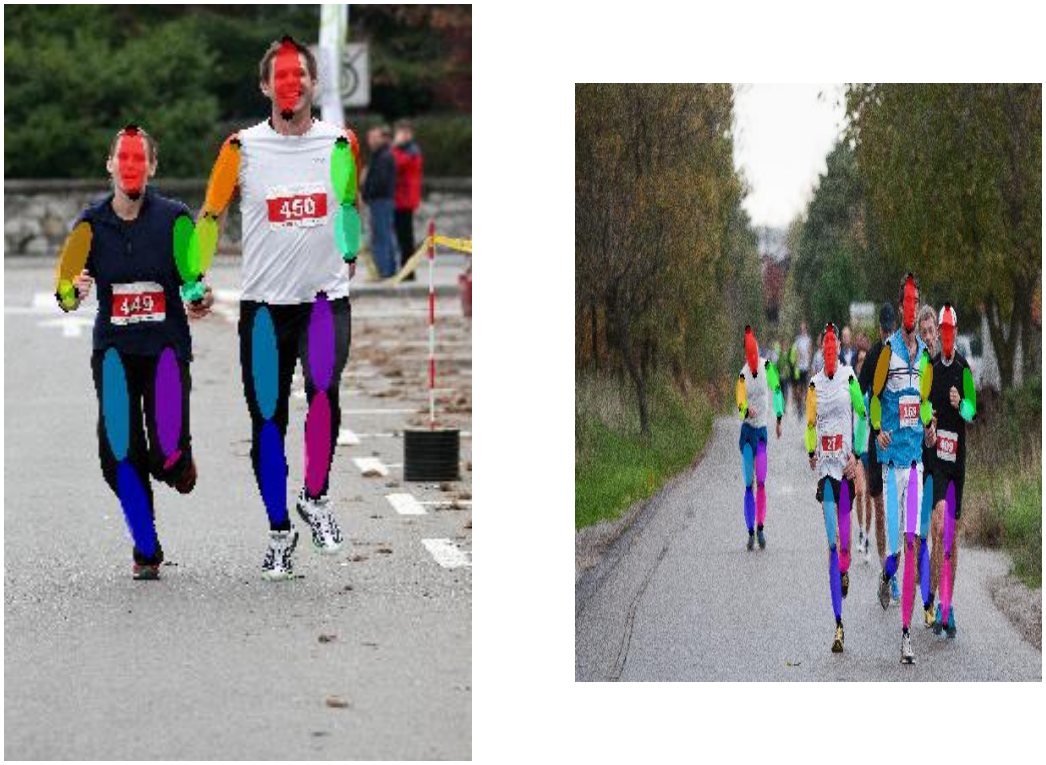
– **Detekcija posameznih delov telesa** Na tem koraku za vsako osebo, z uporabo drugega modela *pose_iter_440000.caffemodel* detektiramo posamezne dele telesa. Metoda vrne koordinate naslednjih delov telesa: glava, vrat, desna rama, desni komolec, desno zapestje, leva rama, levi komolec, levo zapestje, desni bok, desno koleno, desni gleženj, levi bok, levo koleno ter levi gleženj. V repozitoriju so priloženi 4 modeli za prepoznavo delov telesa, vendar na preizkušenih primerih nismo zaznali razlik med njimi. Avtor članka navaja, da na strojni opremi *TitanX* pozamezna detekcija traja 50-65 ms [24]. V kolikor proces parareliziramo, lahko pridemo na 7-9 detekcij na sekundo (brez detekcije oseb).

```
# Model za detekcijo posameznih delov telesa
caffemodel = \
        'lib/Realtime_Multi-Person_Pose_Estimation'
        '/model/_trained_COCO/pose_iter_440000.caffemodel'
```

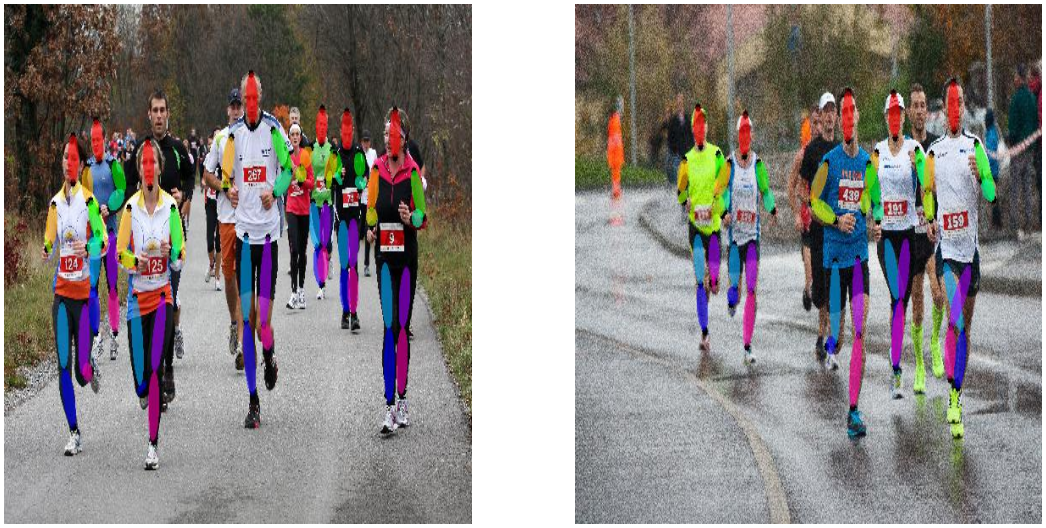


Slika 2.27: Maksimumi verjetnosti zaznave določenega dela telesa.

– **Vizualizacija rezultatov** Na koncu le še povežemo zaznane dele telesa s pomočjo elips različnih barv ter rezultate shranimo v podatkovno bazo za nadaljno obdelavo.



Slika 2.28: Vizualizacija posameznih delov telesa.



Slika 2.29: Vizualizacija posameznih delov telesa.

Časovna zahtevnost

Čas detekcije je močno odvisen od izbrane strojne opreme. Večino spodaj opisanih primerov je bilo izvedenih na prenosnem računalniku s specifikacijami Intel Core i5 2,3 GHz, 4 GB 1333MHz DDR3, Intel HD Graphics 3000 384 MB. Opisana konfiguracija je precej nezmogljiva za izvajanje detekcij. Izvedba celotnega postopka traja približo 60 - 180 sekund. Uporabljene knjižnice obljubljajo procesiranje v *skoraj realnem času*, ob uporabi primerne strojne opreme, torej lahko pričakujemo vsaj 5 slik na sekundo. Nekatere novejše implementacije [25] pa obljubljajo celo procesiranje v *realnem času*.

2.3.4 Primerjava DLIB detektorja obraza ter CNN detektorja pozicije telesa

Tekom razvoja postopka smo preizkusili uporabo različnih detektorjev za detekcijo objektov. Nekateri so dali povsem neprimerne rezultate, zato jih sploh nismo vključil v primerjavo. V *zaključnem izboru*, smo se odločali med uporabo HOG detektorja obraza iz knjižnice DLIB (opisan v sekciji 2.3.2) ter CNN detektorjem pozicije telesa (opisan v sekciji 2.3.3).

Ob primerjavi rezultatov (sekcija 4.2, tabele 4.4 in 4.5), se je CNN detektor, ki detektira pozicijo posameznih delov telesa izkazal za boljšega.

2.4 Selektivno iskanje

Selektivno iskanje je algoritem za segmentacijo slike. Široko je uporabljen, pri detekciji objektov na fotografijah. Njegov namen je iz fotografije izluščiti vse možne regije na sliki, ki morda vsebujejo kakšen objekt. Tako kasneje v postopku detekcije objekta, detektor poženemo le na izračunanih *kandidatih objekta*. Ker je algoritem zelo hiter, je *cenejše* izvesti *selektivno iskanje* in na pridobljenih rezultatih še detektor objekta, kot pa če bi detektor pognali na vseh možnih regijah slike. Zato je osnoven gradnik skoraj vsakega postopka

detekcije objektov.

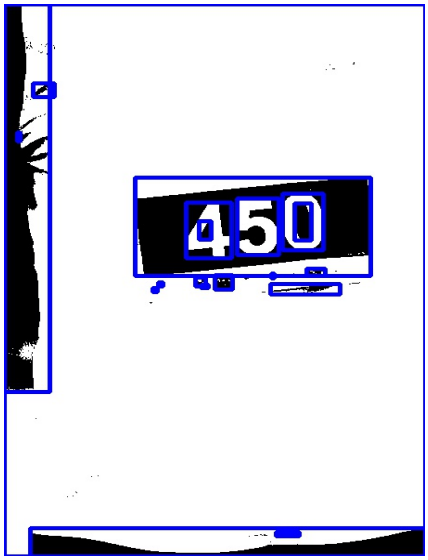
V našem primeru je vhodni podatek algoritmu izolirana regija fotografije, na kateri domnevamo da se nahaja štartna številka (slika 2.30).

Slika 2.30: Tarčne regije definirane v koraku detekcije osebe.

Na tarčnih regijah najprej izvedemo binarizacijo (slika 2.31), saj tako fotografijo spravimo v format, na katerem lahko izvajamo računske operacije. Izkaše se, da najboljše rezultate pridobimo v kolikor uporabljamo z Otsujevo metodo binarizacije (podrobneje opisano v sekciji 2.1).

Slika 2.31: Binarizirane tarčne regije.

Na spletu obstaja več dostopnih implementacij *selektivnega iskanja* algoritma. Uporabil sem Python verzijo knjižnice DLIB [26], ki je sicer napisana v jeziku C++. Metoda za izvedbo algoritma se imenuje *dlib.find_candidate_object_locations()* [27]. Kot argument podamo fotografijo v obliki seznama seznamov. Tak format v tem primeru idealen saj je identičen formatu ki ga uporablja knjižnica OpenCV s katero izvajamo operacije nad fotografijami povsod drugod v postopku. Kot drugi argument prejme spremenljivko *min_size*, ki določa najmanjšo dovoljeno širino rezultatov.



Slika 2.32: Rezultati pridobljeni z metodo dlib.find_candidate_object_locations().

Kot rezultat dobimo tudi veliko manjših oblik, ki ne vsebujejo številke. S pomočjo dodatnih hevrističnih pravil odstranimo večino le-teh.

Argument *min_size* pregleda zgolj širino zaznanega rezultata. Naslednji pogoj pregleda tudi ploščino rezultata.

$$Rezulat = \begin{cases} veljaven, & \text{če } w * h > 200px \\ neveljaven, & \text{sicer} \end{cases} \tag{2.2}$$

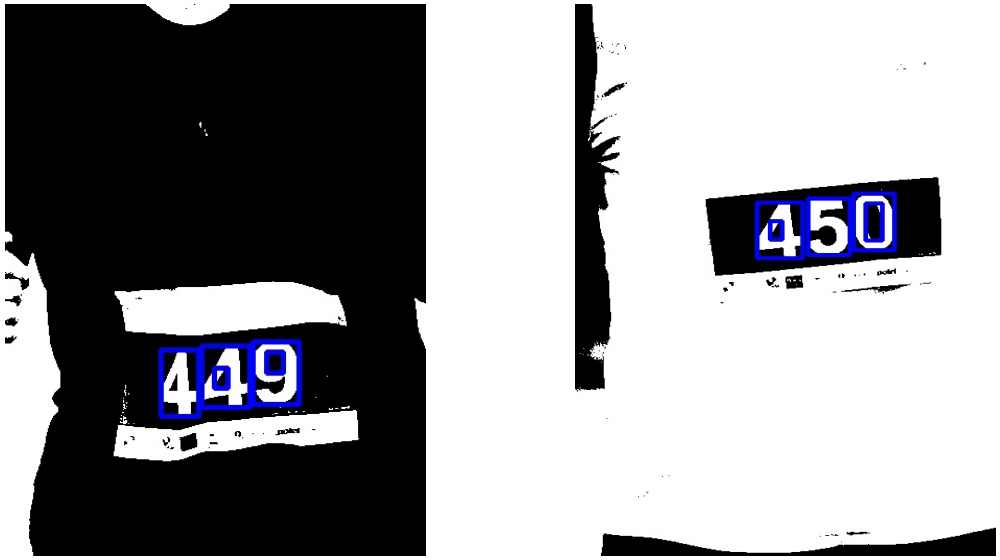
Naslednji pogoj primerja ploščino rezultata s ploščino celotne tarčne regije. Včasih se zgodi da je celotna slika tarčne regije zaznana kot veljaven rezultat. Naslednji pogoj take anomalije izloči (rezultat mora biti za vsaj 70 % manjši od celotne tarčne regije)

$$Rezulat = \begin{cases} veljaven, & \text{če } w * h < (img\_area * 0.3) \\ neveljaven, & \text{sicer} \end{cases} \tag{2.3}$$

Določeni rezultati preprosto ne ustrezajo obliki pravokotnika, ki bi lahko

obdajal številko. Izločimo jih lahko na podlagi razmerja stranic.

$$Rezulat = \begin{cases} veljaven, & \text{če } 0.3 < \frac{w}{h} < 0.9 \\ neveljaven, & \text{sicer} \end{cases} \tag{2.4}$$



Slika 2.33: Filtrirani kandidati objektov.

Preostali rezultati bi morale biti številke z nekaj manjšimi anomalijami za katere bomo poskrbeli pozneje v postopku. V tem koraku posamezne rezultate shranimo kot koordinate (x, y, w, h) oz. kot samostojne fotografije.

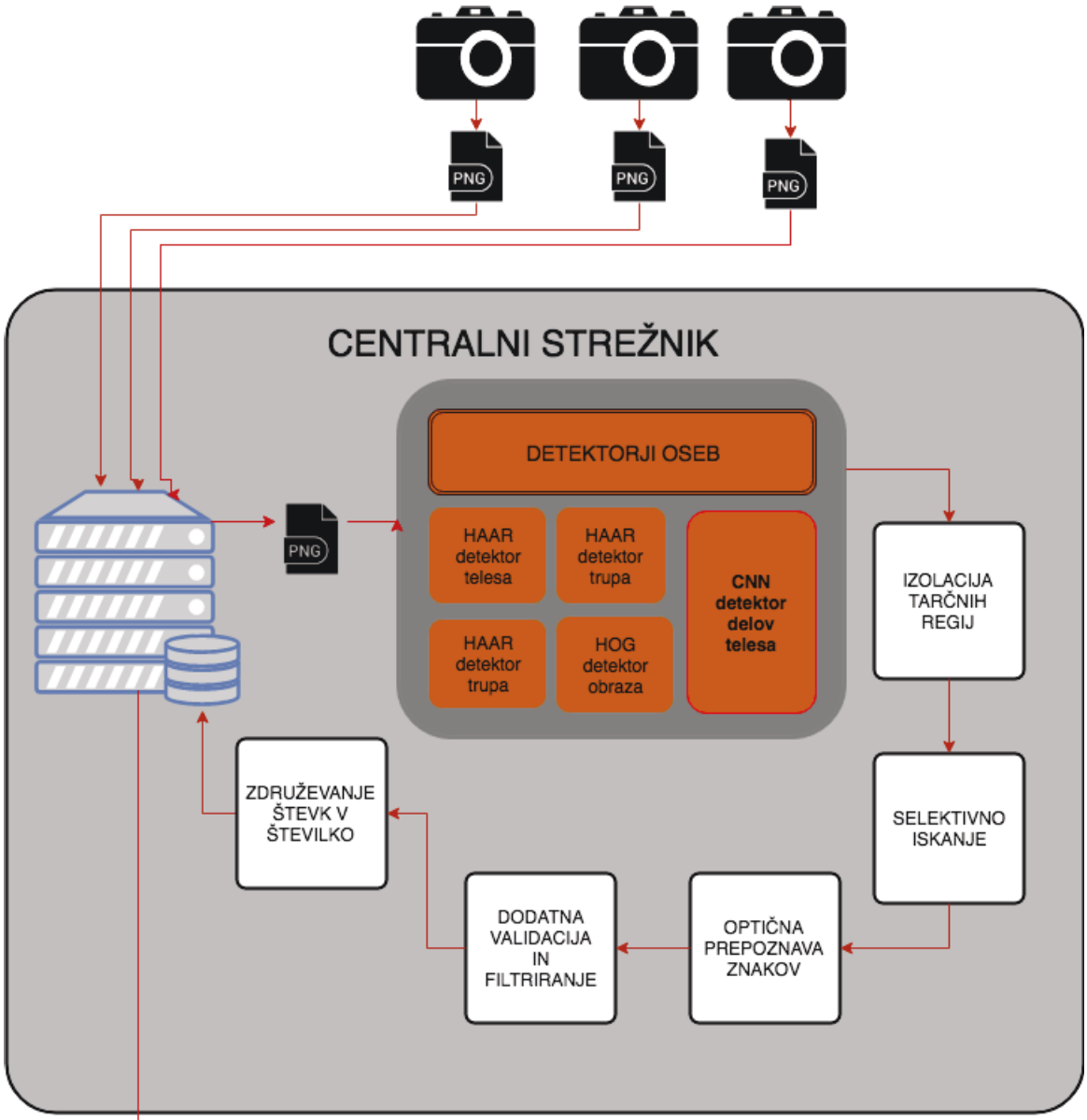
Slika 2.34: Izolirani kandidati objektov.

Poglavje 3

Postopek prepoznave štartne številke

V sledečem poglavju bomo predstavili zaporedje posameznih korakov, ki nas pripeljejo do detekcije štartne številke. Rezultate posameznega koraka bomo tudi vizualno prikazali.

3.1 Shema procesa



ID Fotografije	Zaznana Številka	Verjetnost pravilne prepoznave
#102	433	88%
#102	145	97%
#102	25	20%
#103	13	84%

Slika 3.1: Grafični prikaz procesa prepoznave štartnih številk.

Shema na sliki 3.1 prikazuje celoten proces prepoznave štartne številke. Fotografi, ki so delovali na terenu, fotografije najprej pretočijo na centralni strežnik, kjer se shranijo v primerno strukturo (opisano v sekciji 5.2). Slike iz posameznega dogodka shranimo pod skupno *zbirko*. Nato na vsaki fotografiji iz zbirke izvedemo postopke, ki iz slike prepoznajo štartne številke.

Začnemo z izolacijo tarčnih regij (podrobneje opisano v sekciji 3.2), v katerih pričakujemo, da se nahajajo štartne številke. Za dosego tega najprej uporabimo detektorje različnih delov telesa. Na podlagi dobljenih rezultatov lahko nato izračunamo in izoliramo tarčne regije.

V naslednjem koraku se ukvarjamo z detekcijo posameznih števk, ki se nahajajo v tarčni regiji (podrobneje opisano v sekciji 3.3). Na vsaki izmed regij izvedemo algoritem selektivnega iskanja. Tako dobimo nove izseke slike, na kateri se potencialno nahaja posamezna števka. Na vsakem od njih izvedemo postopek optične prepoznave znakov. Na pridobljenih rezultatih izvedemo dodatno validacijo in filtriranje glede na verjetnost pravilne optične prepoznave, razmerja, lokacije in velikosti znaka.

V zadnjem koraku še posamezne števke združimo v štartno številko (podrobneje opisano v sekciji 3.6).

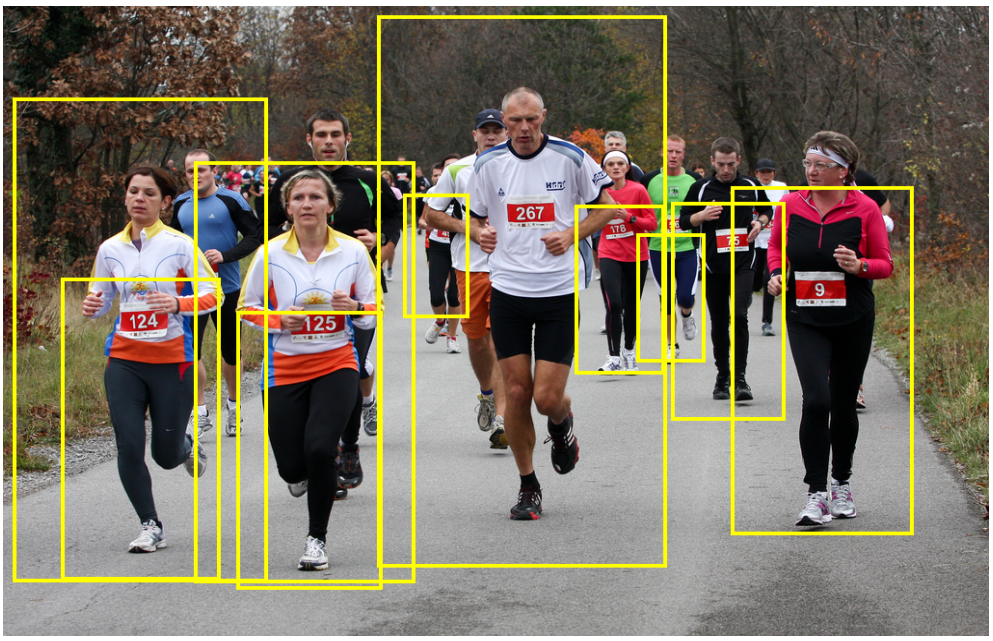
V nadaljevanju bomo posamezen postopek podrobneje opisali.

3.2 Izolacija tarčne regije

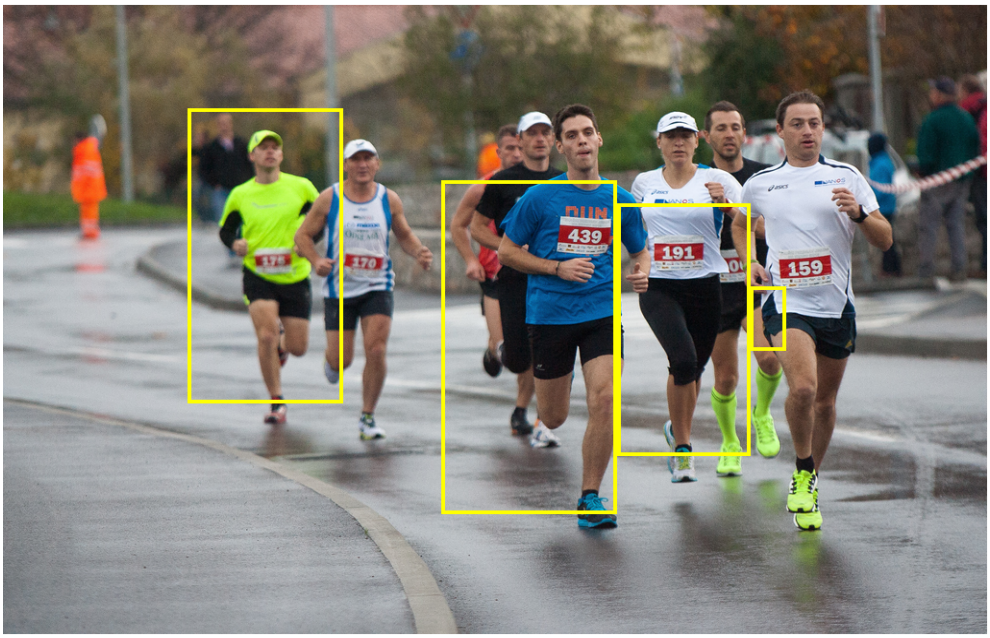
Cilj v prvem koraku postopka je, da izoliramo regije, ki potencialno vsebujejo štartno številko. To lahko storimo na več načinov oz. uporabimo lahko različne detektorje. Nekateri od njih detektirajo obraz, drugi trup, tretji pa posamezne dele telesa. Na podlagi rezultatov omenjenih detektorjev, lahko izračunamo tarčno regijo (anlg. *region of interest*), kjer je največja verjetnost, da se štartna številka nahaja. Na njej bomo nato izvajali nadaljne operacije, ki nas bodo pripeljale do končnih rezultatov. V nadaljevanju bomo opisali nekaj različnih metod oz. detektorjev, ki smo jih implementirali ter preizkusili, za dosego omenjenega cilja.

3.2.1 Detekcija telesa z OpenCV HAAR detektorjem

Klasifikator, ki ga detektor uporablja, je priložen standardni verziji knjižnice OpenCV (*opencv/data/haarcascades/haarcascade_fullbody.xml*). Kadar ga uporabimo na fotografijah tekačev, se izkaže da daje slabe rezultate, saj ne izolira posameznih oseb (slike 3.2 in 3.3). V kolikor je med njimi razmak, klasifikator deluje zadovoljivo, sicer pa ne. Zato je za naš primer povsem neuporaben, saj veliko fotografij vsebuje tekače, ki se delno prekrivajo.



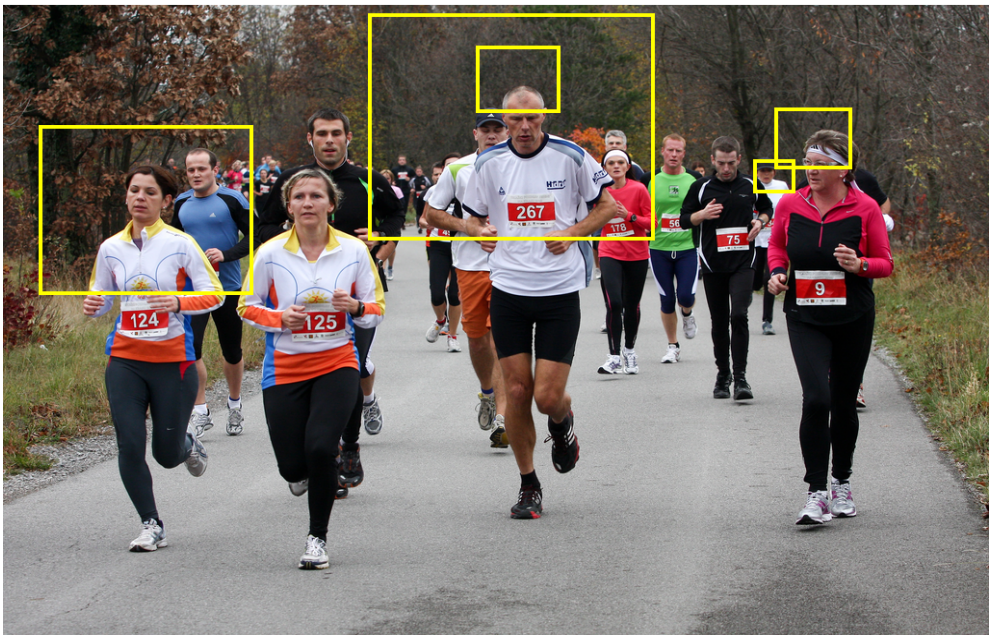
Slika 3.2: Rezultati OpenCV HAAR detektorja z uporabo klasifikatorja *haarcascade_fullbody.xml*.



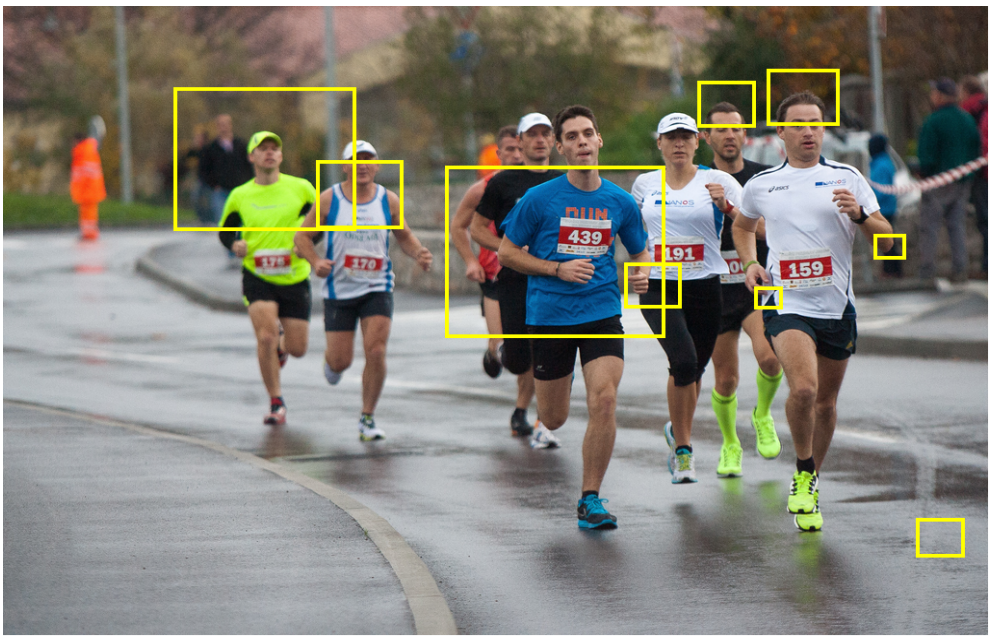
Slika 3.3: Rezultati OpenCV HAAR detektorja z uporabo klasifikatorja *haarcascade_fullbody.xml*.

3.2.2 Detekcija trupa z OpenCV HAAR detektorjem

Klasifikator, ki ga detektor uporablja, je priložen standardni verziji knjižnice OpenCV (*opencv/data/haarcascades/haarcascade_upperbody.xml*). Rezultati so podobni kot pri detektorju *celotnega telesa*, ki je opisan v sekciji 3.2.1. Kot vidimo na slikah 3.4 in 3.5, izolacije oseb praktično ni, hkrati pa rezultati v večini primerov sploh ne zajamejo štartne številke.



Slika 3.4: Rezultati OpenCV HAAR detektorja z uporabo klasifikatorja *haarcascade_upperbody.xml*.



Slika 3.5: Rezultati OpenCV HAAR detektorja z uporabo klasifikatorja *haarcascade_upperbody.xml*.

3.2.3 Detekcija obraza z DLIB HOG detektorjem

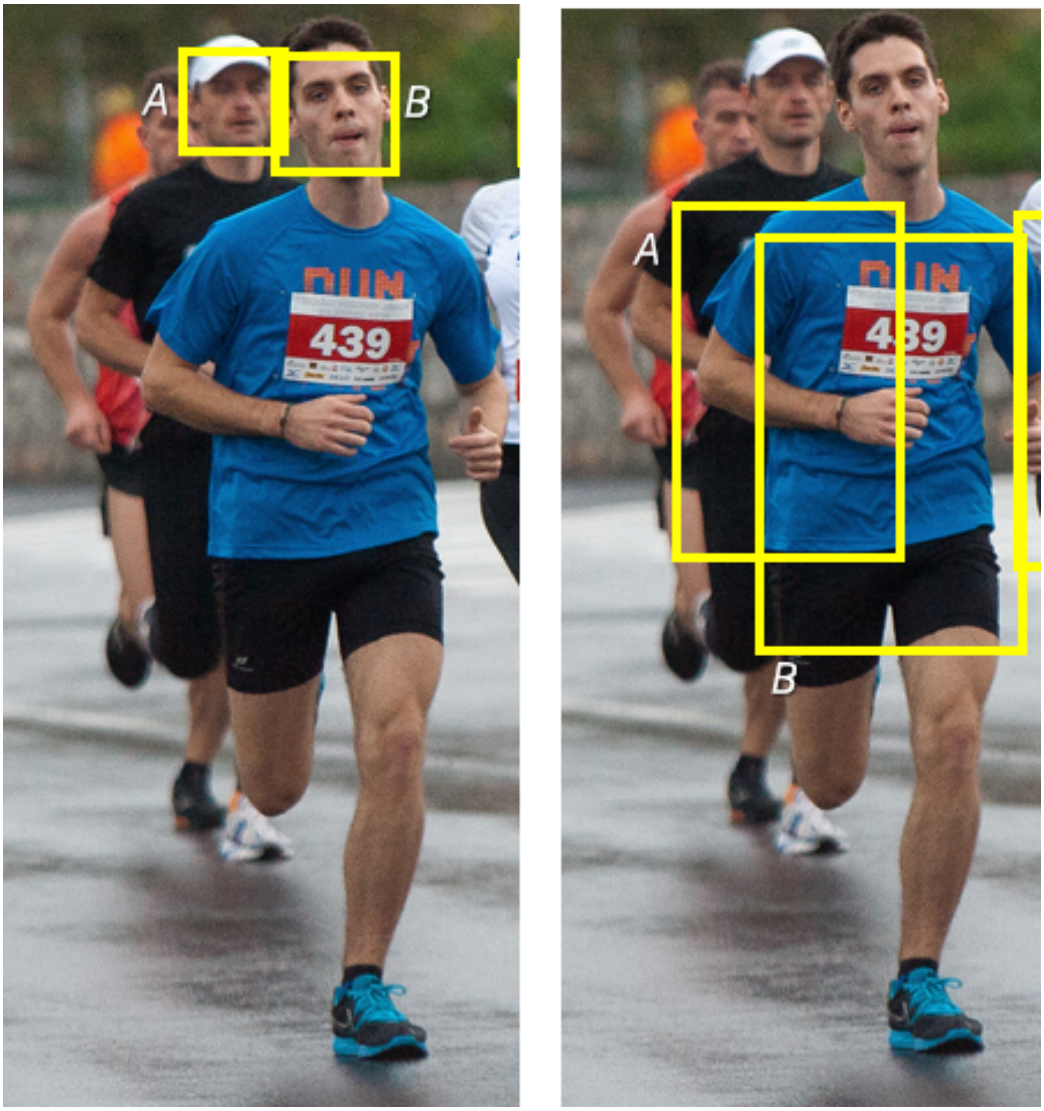
Delno pravilne rezultate pridobimo z uporabo HOG detektorja obraza iz knjižnice DLIB. Detekcija obraza je v primerjavi z detekcijo trupa ali telesa preprostejša za uporabo in daje boljše rezultate. V primerjavi s CNN detektorjem (opisan v sekciji 3.2.4) delov telesa je precej hitrejša in enostavnejša za uporabo, vendar rezultati niso tako natančni.

Za izračun tarčne regije, na podlagi detekcije obraza, izvedemo naslednje operacije:

- Pravokotnik povečamo za faktor 1.2.
- Pravokotnik remaknemo nižje, za višino prvotnega pravokotnika.
- Spodnjo stranico pravokotnika prestavimo nižje za faktor 1.6.

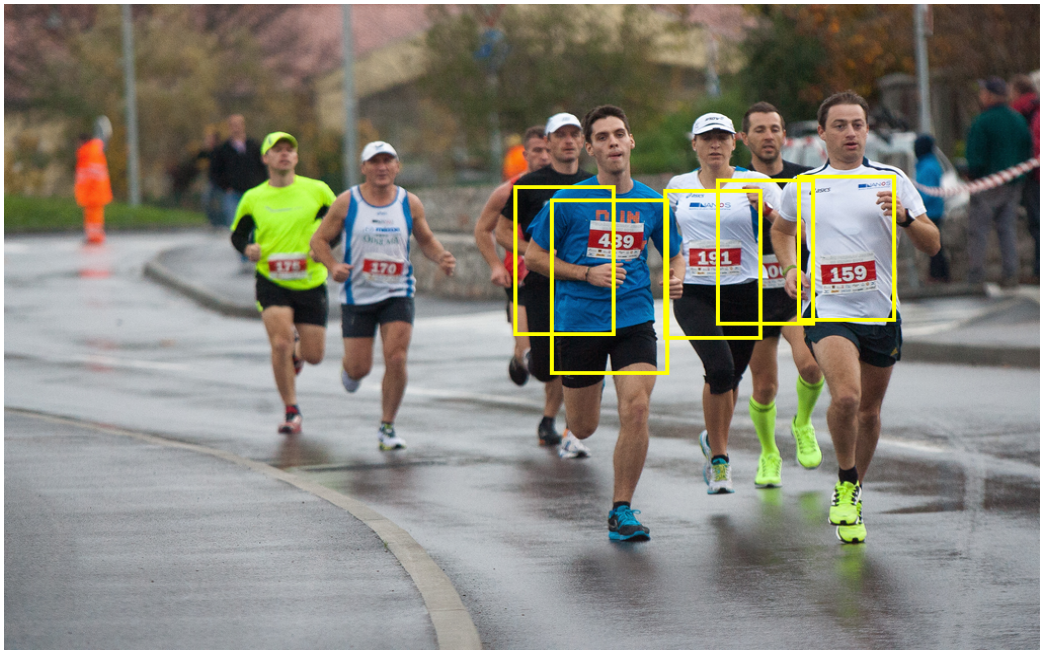
Metoda privede do zadovoljivih rezultatov, vendar se velikokrat zgodi, da izračunana tarčna regija ne pripada pravi osebi.

Kot je prikazano na sliki 3.6, bi bila oseba z obrazom označenim s črko A, prepoznana kot oseba s štartno številko 4 oz. v kolikor bi bilo prekrivanje oseb večje, pa 489. Ta številka identificira osebo B. Podobne težave lahko opazujemo na slikah 3.7, 3.8, 3.9 in 3.10



Slika 3.6: Primer nenatančnega izračuna tarčne regije.

Slika 3.7: Detekcije obraza s knjižnico DLIB.



Slika 3.8: Tarčne regije izračunane na podlagi detekcij obraza.

Slika 3.9: Detekcije obraza s knjižnico DLIB.

Slika 3.10: Tarčne regije izračunane na podlagi detekcij obraza.

3.2.4 Detekcija pozicije telesa z uporabo detektorja delov telesa

Za daleč najboljšega se je izkazal detektor posameznih delov telesa. Dostopen je na portalu *Github.com* pod imenom *convolutional-pose-machines-release*. Kot že samo ime namiguje, detektor uporablja konvolucijske nevronske mreže.

HAAR in HOG detektorji (sekcije 3.2.1, 3.2.2 ter 3.2.3) definirajo tarčno regijo v obliki pravokotnika. V tem primeru dobimo za rezultat množico točk. Vsaka izmed njih definira lokacijo posameznega dela telesa.

V našem primeru povežemo točke *leva rama*, *desna rama*, *desni bok*, *levi bok*, ter *leva rama*. Tako definiramo poligon, ki zelo natančno zaobjema tarčno regijo (slike 3.12 in 3.14). Ta lastnost detektorja je ključna, da so rezultati veliko natančnejši pri HAAR in HOG detektorjih.

Precejšnjo razliko v primerjavi z drugimi detektorji opazimo tudi v sami količini pravilnih detekcij. Lahko bi rekli, da detektor le v redkih primerih zgreši detekcijo samega telesa.

Prekrivanje teles ne predstavlja težave, saj algoritem zakritih teles ne prepozna, saj ne more pozicionirati vseh točk telesa. Tako se v primerjavi z drugimi detektorji znebimo velikega števila lažno pozitivnih rezultatov.

Kot opazimo detektor tudi nima težav z detekcijo teles, ki se nahajajo v *neostrem* delu fotografije (skrajno levi osebi na fotografiji 3.11 oz. skrajno desna detektirana oseba na fotografiji 3.13), vendar so še vedno primerni za detekcijo številke. To je za naš problem ključno, saj so fotografije običajno posnete z objektivi, pri katerih je izostren del fotografije precej majhen.

Tarčno regijo bi lahko dodatno izboljšali tako, da bi za določen faktor povečali spodnjo stranico poligona ter višino poligona. Razlog za to je, da nekateri tekači nosijo štartno številko nekoliko nižje kot običajno oz. nekoliko bolj vstran.

Slika 3.11: Detektirani deli delesa oz. skelet.

Slika 3.12: Tarčna regija definirane na podlagi detektiranih delov telesa.

Slika 3.13: Detektirani deli delesa oz. skelet.

Slika 3.14: Tarčna regija definirane na podlagi detektiranih delov telesa.

3.3 Detekcija posameznih števk v tarčnih regijah

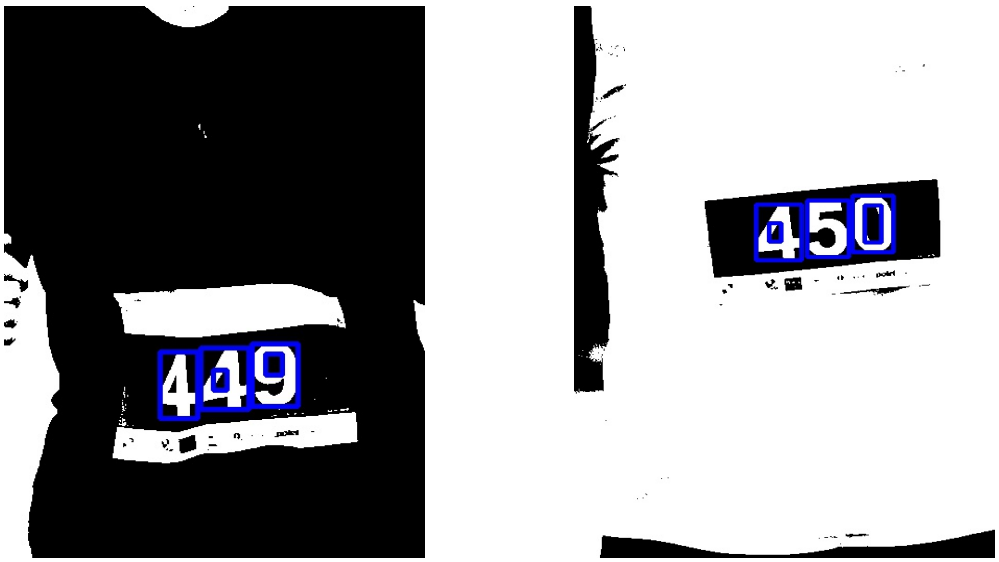
3.3.1 Iskanje kandidatov z uporabo algoritma selektivno iskanje

V tem delu postopka najprej poiščemo t.i. kandidate, na katerih bomo izvedli algoritem, ki bo detektiral ali gre za števko ali ne. Eden od načinov, bi bil pristop z drsečim oknom (*angl. sliding window/kernel*), ki ima enako razmerje stranic kot pravokotnik, ki zaobjema iskano števko. Okno drsi čez sliko in na vsaki regiji izvede funkcijo, ki kot rezultat vrne podatek ali je na sliki števka ter verjetnost pravilnega rezultata. Vsako iteracijo okno horizontalno premaknemo za npr. tretino njegove širine.

Tak pristop si lahko privoščimo, v kolikor je algoritem za prepoznavo števke zares hiter. V našem primeru pa vsebuje dodatne izboljšave za izboljšanje detektiranega rezultata, zaradi katerih nekoliko izgubi na hitrosti. V nekaterih primerih lahko traja do 1 sekunde. Uporaba *drsečega okna* ustvari veliko izsekov, na katerih bi morali izvesti detekcijo številke. Tako bi bil čas izvedbe zelo dolg.

Po opisanem premisleku, se je zdel pristop z uporabo *selektivnega iskanja* veliko boljši, zato smo se omejil na implementacijo le tega.

Za izvedbo *selektivnega iskanja* smo uporabili metodo *dlib.find_candidate_object_locations()* iz knjižnoce DLIB.



Slika 3.15: Primer rezultatov algoritma *Selektivno iskanje*.

Včasih med rezultati ostanejo nekatere nepravilnosti. Lahko bi se jih znebili z uporabo *strožjega filtriranja*, vendar bi s tem tudi izpustili kakšno številko. Na tej točki jih ignoriramo, saj bomo zanje poskrbeli kasneje v postopku. Tako so rezultati v veliki večini primerov res izolirane številke.

Slika 3.16: Izolirani rezultati *Selektivnega iskanja*.

Na predstavljenem primeru (slika 3.16) so na fotografiji številke bele barve, ozadje pa rdeče. Tako po izvedbi postopka binarizacije dobimo slike številk, ki imajo črno ozadje, sama številka pa je bele barve. Na sliki, ki je rezultat binarizacije izvedemo inverz in dobimo posamezne številke zapisane črne na belem ozadju (slika 3.17).

Slika 3.17: Inverz posameznih rezultatov.

3.4 Detekcija števk

Na vsakem rezultatu algoritma *selektivno iskanje*, izvedemo optično prepoznavo znakov. To storimo z orodjem *TesseractOCR*. Izvedemo tudi vse omenjene izboljšave z uporabo rotacije in erozije. Rezultati vsebujejo zaznano števko ter verjetnost pravilne detekcije (tabela 3.1).

Slika	Zaznana Števka	Verjetnost
	1	71 %
	0	93 %
	5	94 %
	4	90 %

Tabela 3.1: Rezultati detekcije posameznih števk

3.5 Napredno filtriranje rezultatov

Na tej točki, imamo za vsako detekcijo dovolj podatkov, da lahko izvedemo *nekoliko pametnejše* filtriranje rezultatov. To storimo predvsem z razlikovanjem rezultatov, ki imajo visoko verjetnost pravilne zaznave, v kombinaciji s preostalimi lastnostmi.

V prvem koraku definiramo števke, ki imajo visoko verjetnost pravilnega rezultata detekcije (v nadaljevanju VVPD). Med njimi so tudi anomalije, ki jih bomo poskušali odstraniti s pomočjo dodatnega filtriranja glede na:

- razliko v višini (sekcija 2.2.1)
- horizontalno oddaljenost od števk z visoko verjetnostjo pravilne detekcije (sekcija 2.2.1)

- vertikalno oddaljenost od števk z visoko verjetnostjo pravilne detekcije (sekcija 2.2.1)

Tabela 3.18 prikazuje rezultate detekcije posameznih števk, ki so bile izolirane iz dela fotografije 3.18. Kot lahko vidimo v tabeli 3.2, smo s pomočjo zgoraj omenjenega filtriranja pravilno izločili napačne rezultate (označeni z rdečo barvo), čeprav je verjetnost pravilnega rezultata precej visoka.

Slika 3.18: Del fotografije na kateri je bil izveden postopek filtriranja rezultatov.

številka	Verjetnost
5	87 %
6	74 %
1	74 %
4	73 %
3	64 %
/	/

Tabela 3.2: Tabela rezultatov in pravilno filtriranih lažno pozitivnih detekcij.

3.6 Združevanje števk v številko

Na koncu števke, ki so prestale proces validacije in so označene kot veljavne (slika 3.19 in tabela 3.3), preprosto združimo v številko na podlagi posameznih x koordinat.

$$x_4 < x_5 < x_0$$

$$4 < 5 < 0$$

Slika 3.19: Primer detektiranih številk.

Slika	Zaznana Številka	Verjetnost	x
	0	93 %	1275
	5	94 %	1226
	4	90 %	1171

Tabela 3.3: Tabela detektiranih številk ter pripadajočih podatkov.

3.7 Izvedba detekcije

Uporabniški vmesnik je namenjen zgolj pregledovanju fotografij ter rezultatov. Celotna detekcija se izvaja z uporabo *višje nivojskih skript*, saj nam omogočajo fleksibilnost ter enostavno in hitro prilagajanje parametrov ter posameznih delov detekcije. Kontrola procesa detekcije preko uporabniškega vmesnika je nesmiselna, saj bi zaradi kompleksnosti za njegov razvoj porabili ogromno časa, hkrati pa bi še vedno bili zelo omejeni in počasni z implementacijo sprememb in novih funkcionalnosti. V nadaljevanju bom opisal

izvedbo celotnega postopka detekcije, ki se je na podlagi rezultatov izkazal za najprimernejšega.

Najprej zaženemo strežnik, ki servira aplikacijo.

```
source virtualenv/bin/activate
python backend/manage.py runserver 127.0.0.1:8080
```

V kolikor je zagon uspešen, se nam izpiše privzeto sporočilo Djangovega razvojnega strežnika

```
Performing system checks...

System check identified no issues (0 silenced).
June 10, 2017 - 16:43:18
Django version 1.10.4, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8080/
Quit the server with CONTROL-C.
```

Prvi korak je, da v sistem uvozimo fotografije na katerih nameravamo izvesti prepoznavo štartnih številk.

```
python tools/import_images.py -i [pot do direktorija s slikami]\
        -n [ime zbirke]

python tools/import_images.py -i samples/mkm_13\
        -n MaliKraskiMaraton2013
```

Naslednji korak je eksekucija detekcije telesa na ciljnih fotografijah.

```
python tools/cnn_pose_estimation.py -i [id fotografije] \
        -c [id zbirke]

python tools/cnn_pose_estimation.py -c 3
```

Rezultati detekcije telesa so shranjeni pod ključem *cnn_pose_body*, izolirane regije trupa pa pod ključem *cnn_pose_torso_est*. V tem koraku poženemo algoritem *selektivno iskanje* na vsaki izmed detekcij *trupa*.

```
python tools/exec_selective_search.py\
        -k [id rezultatov na katerih izvedemo algoritem]\
        -i [id fotografije] -c [id zbirke]

python tools/exec_selective_search.py\
        -k cnn_pose_torso_est -c 3
```

Na vsakem rezultatu *selektivnega iskanja* algoritma poženemo OCR z orodjem Tesseract.

```
python tools/exec_tesseract.py\
        -k [id rezultatov na katerih izvedemo algoritem]\
        -i [id fotografije] -c [id zbirke]

python tools/exec_tesseract.py\
        -k cnn_pose_torso_est_sel_search -c 3
```

Dobljene rezultate validiramo.

```
python tools/validate_tesseract_digit_outputs.py\
        -k [id rezultatov na katerih izvedemo algoritem]\
        -i [id fotografije] -c [id zbirke]

python tools/validate_tesseract_digit_outputs.py\
        -k cnn_pose_torso_est_sel_search -c 3
```

V zadnjem koraku posamezne števke združimo v štartno številko

```
python tools/join_digit_outputs.py\
        -k [id rezultatov na katerih izvedemo algoritem]\
        -i [id fotografije] -c [id zbirke]
```

```
python tools/join_digit_outputs.py\
        -k cnn_pose_torso_est_sel_search -c 3
```

Poglavje 4

Rezultati

4.1 Rezultati optične prepoznave znakov

V tabeli 4.1 so rezultati postopka (izveden na naključnih 28 fotografijah iz zbirke), z uporabo izboljšav optične prepoznave znakov, v tabeli 4.2 pa rezultati brez uporabe le teh. V prvem stolpcu je izpisan identifikator fotografije, drugi stolpec (označen s TP - angl. *true positive*) prikazuje število pravilnih detekcij. Naslednji stolpec (TPR - angl. *true positive rate*) prikazuje stopnjo pravilnih detekcij. Četrti stolpec (FP - angl. *false positive*) prikazuje število napačnih detekcij, zadnji (FN - angl. *false negative*) pa število manjkajočih detekcij.

Zelena barva označuje večje število pravilnih detekcij, rdeča pa majše. Na primer kolikor je v levi tabeli vrstica obarvana z zeleno barvo, pomeni da so zgoraj opisane izboljšave privedle do izboljšave rezultata detekcije.

ID	TP	TPR	FP	FN
#23	3	100 %	0	0
#24	1	100 %	0	0
#25	2	100 %	0	0
#26	1	100 %	0	0
#27	1	50 %	1	1
#28	1	50 %	2	1
#29	2	100 %	1	0
#30	1	33 %	1	2
#31	2	100 %	0	0
#32	0	0 %	3	5
#33	2	100 %	1	0
#34	3	100 %	0	0
#35	1	100 %	0	0
#36	1	100 %	0	0
#37	0	0 %	2	2
#38	3	100 %	0	0
#39	1	100 %	1	0
#40	1	100 %	1	0
#41	2	100 %	0	0
#42	1	100 %	1	0
#43	1	50 %	1	1
#44	1	100 %	0	0
#45	2	100 %	0	0
#46	2	100 %	0	0
#47	0	0 %	1	2
#48	3	75 %	2	1
#49	2	100 %	0	0
#50	1	100 %	0	0

Tabela 4.1: Izvedba postopka z uporabo izboljšav OCR

ID	TP	TPR	FP	FN
#23	1	33 %	2	2
#24	0	0 %	1	1
#25	1	50 %	0	1
#26	0	0 %	1	1
#27	0	0 %	2	2
#28	1	50 %	1	1
#29	1	50 %	1	1
#30	0	0 %	1	3
#31	0	0 %	1	2
#32	0	0 %	1	5
#33	1	50 %	2	1
#34	2	66 %	1	1
#35	0	0 %	1	1
#36	1	100 %	0	0
#37	0	0 %	2	2
#38	1	33 %	1	2
#39	0	0 %	1	1
#40	0	0 %	1	1
#41	2	100 %	0	0
#42	0	0 %	1	1
#43	1	50 %	1	1
#44	0	0 %	1	1
#45	0	0 %	2	2
#46	1	50 %	1	1
#47	0	0 %	0	0
#48	1	25 %	1	3
#49	1	50 %	1	1
#50	1	100 %	0	2

Tabela 4.2: Izvedba postopka brez izboljšav OCR

Z uporabo omenjenih izboljšav smo rezultate detekcije v 20 primerih (71 %) izboljšali, v 8 primerih (29 %) pa ni bilo sprememb.

Kot lahko vidimo v tabeli 4.3 smo rezultate močno izboljšali. V zbirki slik se pojavi 56 štartnih številk, ki jih želimo detektirati. Metoda je uspešno prepoznala (TP) 41 štartnih številk, kar pomeni 73 % uspešnost oz. 55 % vseh detekcij. Zgrešila (FN) jih je 15 (21 %), v 18 (24 %) primerih pa je prepoznala napačno številko.

	TP - pravilne det.	FN - manjkajoče det.	FP - napačne det.
OCR +	41 (55 %)	15 (21 %)	18 (24 %)
OCR -	16 (19 %)	40 (48 %)	28 (33 %)
razlika	+25	-25	-11

Tabela 4.3: Primerjava rezultatov uporabe OCR izboljšav. Vrstica označena z OCR+ vsebuje rezultate z izboljšavami OCR, druga vrstica, ki je označena z OCR- pa rezultate brez njih

4.2 Rezultati detekcije

V tabeli 4.4 so rezultati postopka (izveden na naključnih 28 fotografijah iz zbirke) z uporabo DLIB detektorja obraza, v tabeli 4.5 pa z uporabo CNN detektorja. V prvem stolpcu je izpisan identifikator fotografije, drugi stolpec (TP - angl. *true positive*) prikazuje število pravilnih detekcij. Naslednji stolpec (TPR - angl. *true positive rate*) prikazuje stopnjo pravilnih detekcij. Četrti stolpec (FP - angl. *false positive*) prikazuje število napačnih detekcij, zadnji (FN - angl. *false negative*) pa število manjkajočih detekcij.

Zelena barva označuje boljši rezultat detekcije, rdeča pa slabšega. Na primer v kolikor je v desni tabeli vrstica obarvana z zeleno barvo, pomeni da je uporaba CNN detektorja privedla do boljših rezultatov v primerjavi z uporabo DLIB detektorja obraza.

CNN detektor, ki detektira posamezne dele telesa, se je na izbrani skupini fotografij v 4 primerih (14 %) izkazal za boljšo rešitev, v 2 (7 %) za slabšo,

ID	TP	TPR	FP	FN
#23	2	66 %	0	1
#24	1	100 %	0	0
#25	2	100 %	1	0
#26	1	100 %	0	0
#27	1	50 %	1	1
#28	2	100 %	1	0
#29	2	100 %	1	0
#30	1	33 %	1	2
#31	1	50 %	0	1
#32	3	60 %	2	2
#33	1	50 %	2	1
#34	3	100 %	2	0
#35	1	100 %	1	0
#36	1	100 %	1	0
#37	0	0 %	2	2
#38	2	66 %	1	1
#39	1	100 %	2	0
#40	1	100 %	0	0
#41	2	100 %	0	0
#42	1	100 %	2	0
#43	1	50 %	1	1
#44	1	100 %	0	0
#45	2	100 %	0	0
#46	2	100 %	0	0
#47	0	0 %	1	2
#48	3	75 %	2	1
#49	2	100 %	1	0
#50	1	100 %	0	1

Tabela 4.4: Izvedba postopka z uporabo DLIB detektorja obraza.

ID	TP	TPR	FP	FN
#23	3	100 %	0	0
#24	1	100 %	0	0
#25	2	100 %	0	0
#26	1	100 %	0	0
#27	1	50 %	1	1
#28	1	50 %	2	1
#29	2	100 %	1	0
#30	1	33 %	1	2
#31	2	100 %	0	0
#32	0	0 %	3	5
#33	2	100 %	1	0
#34	3	100 %	0	0
#35	1	100 %	0	0
#36	1	100 %	0	0
#37	0	0 %	2	2
#38	3	100 %	0	0
#39	1	100 %	1	0
#40	1	100 %	1	0
#41	2	100 %	0	0
#42	1	100 %	1	0
#43	1	50 %	1	1
#44	1	100 %	0	0
#45	2	100 %	0	0
#46	2	100 %	0	0
#47	0	0 %	1	2
#48	3	75 %	2	1
#49	2	100 %	0	0
#50	1	100 %	0	0

Tabela 4.5: Izvedba postopka z uporabo CNN detektorja pozicij telesa.

v 22 primerih (79 %) pa ni bilo sprememb.

Kot vidimo v tabeli 4.6 je število pravilnih detekcij ostalo enako (73 % uspešnost). Z uporabo CNN detektorja, smo dobili bolj natančno definicijo tarčne regije in s tem zmanjšali število manjkajočih detekcij za 7.

Potrebno je omeniti, da je uporaba DLIB detektorja obraza povzročila daljši čas celotnega postopka, saj metoda definira večjo tarčno regijo. Tako v naslednjem koraku selektivno iskanje vrne veliko več regij, kot pri uporabi CNN detektorja.

	TP - pravilne det.	FP - napačne det.	FN - manjkajoče det.
CNN	41 (55 %)	18 (24 %)	15 (20 %)
DLIB	41 (51 %)	25 (31 %)	15 (18 %)
razlika	0	-7	0

Tabela 4.6: Primerjava rezultatov uporabe DLIB detektorja obraza in CNN detektorja pozicije telesa.

4.3 Rezultati identifikacije

V tabeli 4.7 so predstavljeni končni rezultati postopka, ki je podrobneje opisan v poglavju 3. Prvi stolpec prikazuje število pravilnih detekcij (TP), drugi stolpec prikazuje število napačnih detekcij, Zadnji stolpec pa prikazuje število manjkajočih detekcij.

Zbirka na kateri smo pognali postopek, vsebuje rezultate detekcije iz 65 fotografij. 20 od njih smo uporabljali med samim razvojem postopka, ostalih 45 je novih. Fotografije so naključno izbrane iz različnih tekaških dogodkov v Sežani (*Mali kraški maraton, Tekaški pozdrav jeseni*). Na večini izmed njih so 2 - 3 osebe, na nekaterih je do 10 oseb, na drugih pa samo 1. Večina je posneta v ležeči kompoziciji, nekatere izmed njih pa v pokončni. Na fotografijah se pojavi 145 štartnih številk, metoda jih je prepoznala 85 kar pomeni 58 % uspešnost oz. 43 % vseh detekcij. 53 detekcij je prepoznala za napačne številke (27 %), izpustila pa je 60 štartnih številk (30 %).

TP - pravilne det.	FP - napačne det.	FN - manjkajoče det.
85 (43 %)	53 (27 %)	60 (30 %)

Tabela 4.7: Rezultati detekcije na testni zbirki fotografij.

Rezultati so se precej poslabšali v primerjavi z rezultati, ki smo jih pridobili z izvedbo postopka na zbirki slik, ki smo uporabljali pri razvoju. Tipične napake so:

- Detektor oseb ne detektira pozamezne osebe, torej se za dano osebo tarčna regija sploh ne izračuna ter optična prepoznava znakov sploh ne izvede.
- Zaradi različnih barv tekaških majic, metoda binarizacije Otsu nepravilno binarizira sliko. Posledica tega je, da algoritem selektivno iskanje ne izolira posameznih števk za kasnejšo prepoznavo.
- V primerih ko je na sliki oseba delno zakrita (pred njo stoji druga oseba) niso vidne vse števke, ki sestavljajo štartno številko. Tako sistem prepozna samo nekaj izmed njih (podrobneje opisano v sekciji 6.4).

Tudi če rezultati na prvi pogled izgledajo slabi, še vedno aplikacija opravi svojo nalogo. Posamezniku veliko množico slik zmanjša na samo nekaj fotografij, na kateri je bila detektirana njegova štartna številka. Vmes se seveda zaradi napačnih detekcij pojavijo tudi slike na katerih te osebe ni, vendar je teh slik še veliko manj v primerjavi s številom vseh slik v galeriji.

Poglavje 5

Uporabniški vmesnik in organizacija fotografij

5.1 Uporabniški vmesnik

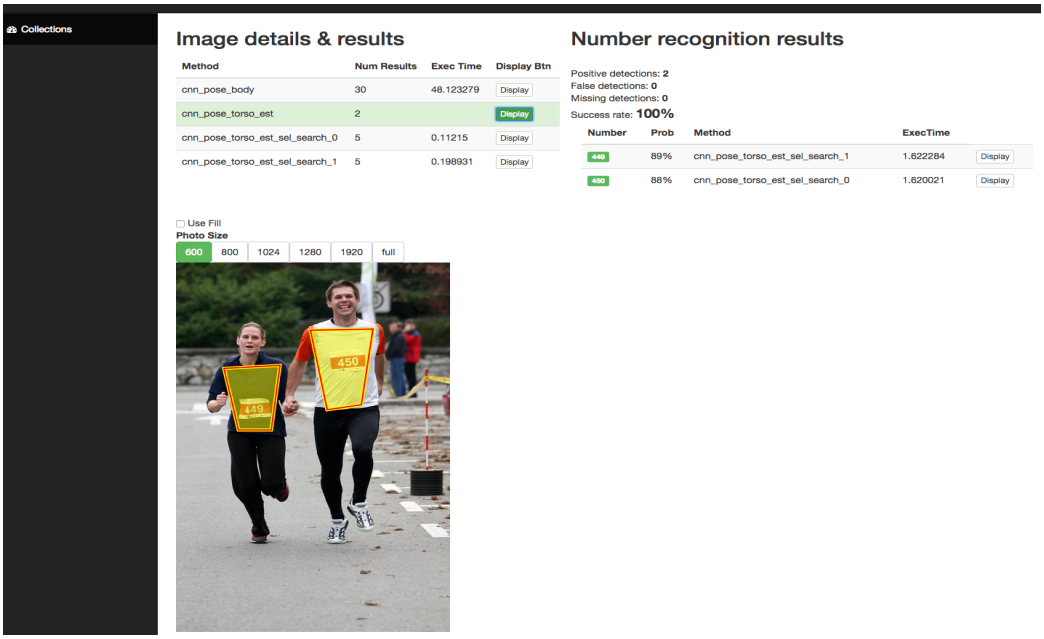
Implementacija uporabniškega vmesnika ni bila primaren cilj diplomske naloge, vendar bi brez njega težko prišli do opisanih rezultatov. Sama implementacija opisanega postopka je obsegala veliko testiranja in prilagajanja parametrov. Brez dobre, hitre in efektivne vizualizacije to preprosto ne bi bilo mogoče. Hkrati je potrebno poudariti, da je uporabniški vmesnik namenjen zgolj prikazu fotografij in vizualizaciji rezultatov posameznih algoritmov.

Kot lahko vidimo na sliki 5.1, so osnovni gradniki vmesnika:

- tabela v kateri lahko izbiramo med rezultati posameznih metod, ki bodo sočasno vizualizirani na fotografiji (na sliki levo zgoraj)

- tabela končnih rezultatov, ki prikazuje katere številke smo pravilno detektirali (na sliki desno zgoraj)

- vizualizacija rezultatov na fotografiji, kjer lahko poljubno izbiramo velikost fotografije (na sliki levo spodaj)



Slika 5.1: Vizualizacija in izbira prikazanih rezultatov na dani fotografiji

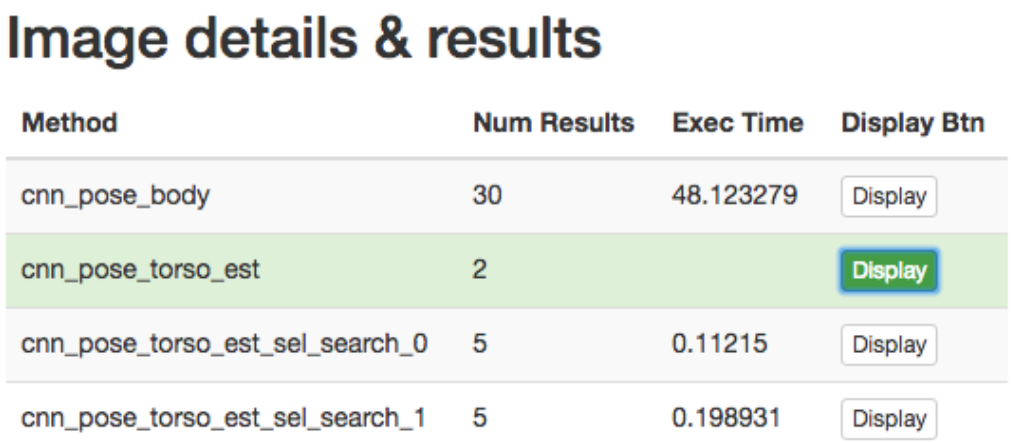
Vmesni rezultati postopka

S pomočjo tabele, prikazane na sliki 5.2, lahko vidimo kateri rezultati so na voljo na posamezni fotografiji, koliko elementov vsebujejo, ter koliko časa je potekal izračun oz. detekcija. Ob kliku na gumb, izbrane rezultate vizualiziramo na fotografiji.

Iz tabele lahko razberemo, da v kolikor kliknemo na gumb *Display* v prvi vrstici, bomo na fotografiji vizualizirali 30 rezultatov metode *cnn_pose_body*, ki vsebuje točke posameznih delov teles. Izvedba omenjene metode je trajala 48 sekund.

Druga vrstica prikazuje 2 izračunani tarčni regiji (*cnn_pose_torso_est*).

Tretja in četrta vrstica vsebujeta rezultate selektivnega iskanja na posamezni od izračunanih tarčnih regij. Kot vidimo je v obeh primerih selektivno iskanje vrnilo 5 rezultatov, čas izvedbe pa je bil med 0,1-0,2 sekunde.



Slika 5.2: Tabela vmesnih rezultatov.

Zaznane števke

Tabela na sliki 5.3 prikazuje katere števke so bile zaznane na fotografiji. Izpisana je zaznana števka (*Number*), verjetnost pravilne zaznave (*Confidence*), koordinate števke na fotografiji (*Location*), velikost zaznane števke (*Rect Size*), ploščina zaznane števke (*Rect Area*), koliko sekund je orodje Tesseract porabilo za zaznavo (*TesseractTime*), ter rezultati validacije (*Valid*).

V zadnji vrstici tabele vidimo, da je bila številka 1 zaznana z 71 % verjetnostjo pravilne prepoznave. V zadnjem stolpcu, pa je prikazano da je bil rezultat označen kot *lažno pozitiven*, saj je razlika v višini števke v primerjavi s števkami z visoko verjetnostjo pravilne detekcije prevelika (43 %, maksimalna dovoljena pa je 6,3 %).

Digit Recognitions

Digit	Confidence	Location	Rect Size	Rect Area	TesseractTime	Height Diff - Valid
4	93%	X: 538 Y: 1219	46 x 61	2806	1.278457	OK
9	92%	X: 587 Y: 1215	44 x 62	2728	1.252722	OK
4	88%	X: 499 Y: 1223	36 x 66	2376	1.271895	OK
1	71%	X: 550 Y: 1241	13 x 20	260	1.287985	H diff 43.0% (max allow: 6.3%)

Slika 5.3: Tabela zaznanih števk.

Končni rezultati detekcije

S pomočjo tabele na sliki 5.4 lahko vidimo končne rezultate celotnega postopka. Prikazuje zaznane številke na fotografiji. Izpisana je zaznana številka (*Number*), verjetnost pravilne detekcije (*Prob*), ime izolirane tarčne regije (*Method*), porabljen čas za izvedbo detekcije številke (*ExecTime*) ter nekaj osnovnih podatkov o uspešnosti zaznave, v kolikor imamo vnaprej definirane pravilne rezultate.

Number recognition results

Positive detections: **2**
False detections: **0**
Missing detections: **0**
Success rate: **100%**

Number	Prob	Method	ExecTime	
449	89%	cnn_pose_torso_est_sel_search_1	1.622284	Display
450	88%	cnn_pose_torso_est_sel_search_0	1.620021	Display

Slika 5.4: Tabela končnih rezultatov postopka.

Prikaz fotografije in izbranih rezultatov

V spodnjem delu vmesnika (slika 5.5), lahko izbrane vmesne rezultate vizualiziramo na fotografiji. S pomočjo gumbov lahko izbiramo velikost prikazane fotografije, saj je fotografija v polni velikosti navadno prevelika za prikaz na ekranu.

Slika 5.5: Fotografija ter vizualizacija izbranih rezultatov.

5.2 Podatkovni model

Prvi korak v celotnem postopku je uvoz fotografij v sistem. Ob tem jih moramo shraniti na tak način, da bomo lahko nad njimi efektivno izvajali nadaljne operacije ter shranjevali rezultate.

Kot primer lahko navademo izvedbo ukaza

```
python tools/exec_selective_search.py -k cnn_pose_torso_est -c 3
```

Iz podatkovne baze se bodo najprej naložile vse fotografije, ki so v zbirki (*collection*) z identifikatorjem 3. Na vsaki izmed slik bodo v atributu *results* poiskani rezultati shranjeni pod ključem *cnn_pose_torso_est*. Na njih se bo izvedla metoda selektivnega iskanja. Rezulati metode bodo shranjeni pod nov ključ.

Vsi postopki omenjeni v sekciji 3.7 se izvajajo na podoben način. Spodaj opisani modeli so definirani na način, da omogočajo tako izvedbo.

Podatkovni model definira 2 osnovna razreda. Model Zbirka (angl. Collection) predstavlja skupino fotografij. V praksi bi vse fotografije iz danega dogodka združili v eno zbirko.

Model Slika (angl. Image) pa vsebuje pot do datoteke fotografije, nekaj meta podatkov ter dodatne atribute, v katere shranjujemo vmesne rezultate ter tarčne regije. Končne rezultate zaznav shranimo v atribut *outputs*.

```
class Collection(Model):
    '''
    Preprost model za grupiranje fotografij.
    V prihodnosti bi lahko dodali
        posamezne atribute za potrebne informacije
    '''
    name = CharField(max_length=200)


class Image(Model):
    '''
        Podatkovni model ki predstavlja vsako fotografijo
    '''
    # datoteka fotografije
    file = FileField(upload_to=image_path)
    # pomanjšana različica fotografije za
    # hitro prikazovanje v seznamih (thumb)
    thumb_file = FileField(upload_to=image_thumb_path, null=True)
    # Relacija ki pove kateri zbirki (Collection)
    # pripada fotografija
    collection = ForeignKey(Collection,
        related_name='images', on_delete=CASCADE)
    # JSON atribut v katerega shranjujemo vmesne rezultate
    results = JSONField(null=True)
    # Prirejena kopija rezultatov, preračunanih za prikazovanje
    # na manjših različicah fotografije
    results_scaled = JSONField(null=True)
    # Ročno vnešeni rezutati, ki jih pričakujemo.
    # Z uporabo tega atributa lahko
    # izračunamo uspešnost zaznave
    test_results = JSONField(null=True)
    # Končni rezultati zaznave. Posamezne števke, številke...
    outputs = JSONField(null=True)
    # Shranjena širina fotografije
    width = IntegerField(null=True)
    # Shranjena višina fotografije
    height = IntegerField(null=True)
```

Slika 5.6: Koda definicije podatkovnega modela.

Poglavje 6

Ideje za nadaljni razvoj in nerešeni problemi

6.1 Dodatna validacija z detekcijo starosti in spola

Postopek se fokusira na detekcijo štartne številke, celoten cilj detekcije pa je identifikacija oseb na fotografijah. Za dosego boljših rezultatov bi lahko implementirali dodatne postopke identifikacije.

V naslednjih primerih predpostavljamo, da imamo dostop do osebnih podatkov o udeležencih prireditev kot so starost, spol, fotografije.

6.2 Validacija rezultatov na podlagi ocene starosti in spola z uporabo nevronskih mrež

Kot dodatno validacijo rezultatov razvitega postopka, bi lahko izvedli detekcijo starosti in spola oseb, kjer smo zaznali štartno številko. Tako bi lahko dodatno izločili morebitne lažne pozitivne prepoznave.

Tudi v tem koraku lahko uporabimo konvolucijske nevronske mreže. Zaradi preprostosti implementacije, bi uporabil orodje Keras ter primer za-

znave, ki je prikazan v Githubovem repozitoriju *age-gender-estimation* [28].

Postopek najprej uporabi detektor obraza iz knjižnice DLIB (opisan v sekciji 3.2.3) za lokalizacijo obrazov. V naslednjem koraku na vsakem obrazu izvede detekcijo starosti ter spola, z modelom ki ga je ustvaril avtor članka. V našem primeru bi lahko obraz locirali tudi na podlagi detekcije lokacije telesa.

Na sliki 6.1 lahko vidimo, da je model detektiral 2 moška stara 27 let. Pravilen rezultat bi bil ženska 32 let ter moški 35 let. Detektor smo preizkusili tudi na množici drugih fotografij. Rezultati so se izkazali za premalo natančne, da bi bili uporabni.

Slika 6.1: Primer napačne prepoznave spola in starosti.

6.3 Primerjava obrazov

Ob predpostavki da dostopamo do uporabniških fotografij (npr. z omrežja Facebook), bi lahko kreirali model za prepoznavo obraza, ki bi obraze na fotografijah primerjal z obrazi iz kreirane zbirke. Tako bi lahko dodatno izboljšali pravilnost rezultatov ter morda celo detektirali določene rezultate, ki smo jih izpustili zaradi napake v detekciji pozicije telesa.

Eno enostavnejših odprtokodnih orodij za implementacijo najdemo v Github-hubovem repozitoriju *ageitgey/face_recognition*[29]. Orodje uporablja knjižnico DLIB (ki lahko uporablja nevronske mreže, v kolikor skonfiguriramo nekaj dodatnih programskih paketov).

Klasifikatorje bi lahko kreirali že pred dogodkom, nato pa zgolj pognali detektor.

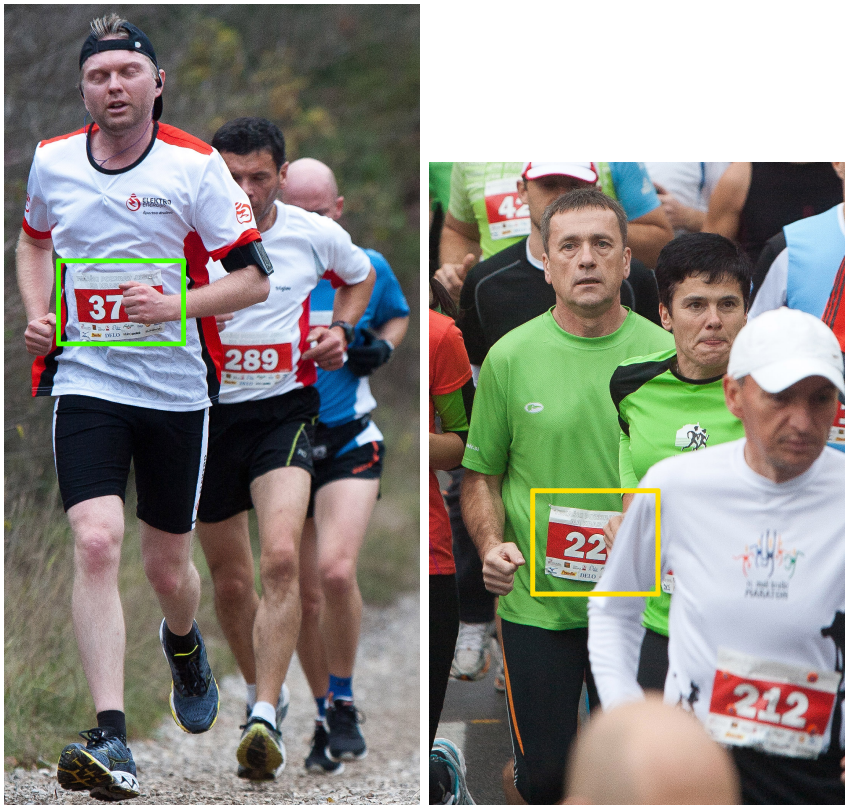
```
import face_recognition
known_image = face_recognition.load_image_file("znana_oseba_1.jpg")
unknown_image = face_recognition.load_image_file("neznana_oseba.jpg")

# Kreiranje klasifikatorjev
known_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

# Prepoznava oseb
results = face_recognition.compare_faces(
        [known_encoding], unknown_encoding)
```

Slika 6.2: Primer preprostega kreiranja detektorja ter prepoznave obrazov.

6.4 Problem prekritih številk

Eden izmed težjih nerešenih problemov, so prekrite številke. Kot lahko vidimo na sliki 6.3, se velikokrat zgodi, da eno izmed številk, ki sestavljajo štartno

številko, prekriva drugi tekač oz. tekačeva roka.



Slika 6.3: Primeri fotografij, ki vsebujejo prekrite številke.

Številka na levi fotografiji slike 6.3, bi bila prepoznana kot 37, na desni fotografiji pa 22. V obeh primerih je dan rezultat napačen, saj se sistem ne zaveda skrite številke, ki se je ne da detektirati.

Enostavna rešitev bi bil dogovor z organizatorjem, da uporablja n-mestne številke. V primeru da se dogodka udeleži 1000 tekačev, bi uporabili 4 mestne številke. Številko 1, bi torej zapisali kot 0001. V kolikor prepoznana številka vsebuje manj kot n številk, vemo da je rezultat neveljaven. Ta rešitev bi bila tehnološko zelo enostavna, vendar večina organizatorjev, takega kompromisa ne bi sprejela.

Druga možna rešitev bi bili markerji. Vsak štartni listek bi vseboval 4 markerje - vsakega v enem od kotov štartne številke. Markerji so grafike,

ki jih enostavno prepoznamo s programskim orodjem. V kolikor na prepoznani številki ne moremo *prešteti* vseh štirih markerjev, rezultat označimo kot neveljaven. Problem tega pristopa je, da bi najverjetneje zavrgli tudi veliko pravilnih rezultatov. Hkrati moramo z organizatorjem uskladiti obliko štartne številke, da bi vsebovala markerje, kar spet ni idealna rešitev.

Naslednja možna rešitev bi bilo preverjanje razmerja stranic pravokotnika, ki zaobjema številke. V kolikor pravokotnik ne ustreza pravemu razmerju, rezultat zavržemo, saj je velika verjetnost da je vsaj ena izmed številk prekrita. Tak pristop daje dobre rezultate. V primeru da so številke na unikatnem ozadju (kot na primer rdeče odzadje na sliki 6.3). V kolikor številke nimajo enakomernega ozadja, pristop ne deluje več. Tudi v tem primeru bi bilo potrebno usklajevanje z organizatorjem glede oblike štartne številke.

Poglavje 7

Zaključek

V diplomski nalogi smo najprej predstavili motivacijo in problem identifikacije tekačev na fotografijah. Ugotovili smo, da je najprimernejša metoda identifikacije osebe, da prepoznamo štartno številko, ki jo vsak tekač nosi na dresu. To smo dosegli tako, da smo najprej s pomočjo različnih detektorjev objektov iz fotografije izolirali posamezne osebe. Ugotovili smo, da se je najbolj izkazal detektor posameznih delov telesa, ki uporablja konvolucijske nevronske mreže. Na podlagi dobljenih rezultatov smo zlahka definirali tarčno regijo v kateri se nahaja štartna številka. Nato smo na njej izvedli algoritem selektivnega iskanja, ki nam je definiral izseke slik, na katerih se morda nahaja posamezna številka. Na vsakem izseku smo s pomočjo orodja *TesseractOCR* izvedli optično detekcijo znakov. Vse rezultate smo dodatno validirali in filtrirali. Preostale rezultate smo le še združili v celo štartno številko.

V diplomskemu delu smo tudi podrobno predstavili uporabljene metode in orodja. Predstavili smo delovanje različnih detektorjev objektov. Razložili smo vse uporabljene izboljšave algoritma selektivno iskanje ter optične detekcije znakov. Opisali smo tudi vsa orodja, ki smo jih tekom postopka uporabljali za obdelavo fotografij.

Razvili smo sodoben uporabniški vmesnik, ki je služil pregledovanju fotografij in vizualizaciji rezultatov. Z njegovo pomočjo smo algoritme in različne

korake postopka veliko lažje prilagajali, saj smo lahko na pregleden način hitro opazili razlike.

Na koncu smo predstavili, kako smo si v ozadju aplikacije organizirali fotografije ter kako izgleda praktična uporaba razvitega postopka v ukazni vrstici. Pokazali smo tudi nekaj idej, ki so se izkazale za neuspešne ter opisali probleme, ki jih nismo uspeli rešiti tekom razvoja diplomske naloge.

Diplomska naloga nam je predstavljala velik izziv. Vseskozi smo imeli v mislih kako bi razvili funkcionalen sistem in ne zgolj prototipa za namen realizacije diplomske naloge. Tekom razvoja smo se veliko naučili, še ogromno pa ostaja neznanega. Iskreno upamo, da bomo imeli čas in priložnost sistem še nekoliko izpopolniti ter ga preizkusiti v produkcijskem okolju.

Literatura

[1] R. Smith. (2014) Tesseract ocr. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: https://github.com/tesseract-ocr/

[2] Google. (2017) Vision api - image content analysis — google cloud platform. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: https://cloud.google.com/vision/

[3] openalpr. (2013) Automatic license plate recognition library. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: https://github.com/openalpr/openalpr

[4] F. Niklas. (2017) Binarization. [Dostopano: 21.8.2017]. [Online]. Dosegljivo: http://felixniklas.com/imageprocessing/binarization

[5] (2016) Image thresholding. [Dostopano: 21.8.2017]. [Online]. Dosegljivo: http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html

[6] T. Breuel. (2016) hocr-spec. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: https://github.com/kba/hocr-spec

[7] Wikipedia. (2017) hocr - wikipedia. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/HOCR

[8] (2017) Eroding and dilating. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[9] (2017) Face detection using haar cascades. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[10] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–511–I–518 vol.1.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, ser. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. [Online]. Dosegljivo: http://dx.doi.org/10.1109/CVPR.2005.177

[12] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part II*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 428–441. [Online]. Dosegljivo: http://dx.doi.org/10.1007/11744047_33

[13] N. Corporation. (2017) Cuda zone — nvidia developer. [Dostopano: 25.9.2017]. [Online]. Dosegljivo: https://developer.nvidia.com/cudnn

[14] T. Dettmers. (2017) Which gpu(s) to get for deep learning: My experience and advice for using gpus in deep learning. [Dostopano: 22. 6. 2017]. [Online]. Dosegljivo: http://timdettmers.com/2017/04/09/which-gpu-for-deep-learning/

[15] iomart Group plc. (2017) Gpu server hosting — nvidia gpu servers. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: http://www.redstation.com/gpu-server

[16] OVH. (2017) Dedicated servers with graphics cpu - gpu - ovh. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: https://www.ovh.com/us/dedicated-servers/gpu/

[17] Hetzner. (2017) Dedicated root server hosting - hetzner online gmbh. [Dostopano: 4.8.2017]. [Online]. Dosegljivo: https://www.hetzner.com/dedicated-rootserver/ex51-ssd-gpu?country=sk

[18] BCLC. (2017) Windows caffe. [Dostopano: 25.9.2017]. [Online]. Dosegljivo: https://github.com/BVLC/caffe/tree/windows

[19] N. Corporation. (2017) Cuda zone — nvidia developer. [Dostopano: 25.9.2017]. [Online]. Dosegljivo: https://developer.nvidia.com/cuda-zone

[20] S. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," *CoRR*, vol. abs/1602.00134, 2016. [Online]. Dosegljivo: http://arxiv.org/abs/1602.00134

[21] T. K. Y. S. Shih-En Wei, Varun Ramakrishna. (2016) Code repository for convolutional pose machines, cvpr'16. [Dostopano: 24.8.2017]. [Online]. Dosegljivo: https://github.com/shihenw/convolutional-pose-machines-release/

[22] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[23] B. Sapp and B. Taskar, "Modec: Multimodal decomposable models for human pose estimation," in *In Proc. CVPR*, 2013.

[24] shihenw. (2016) Convolutional pose machines (for multi-person) python demo code. [Dostopano: 22. 6. 2017]. [Online]. Dosegljivo: https://github.com/shihenw/convolutional-pose-machines-release/blob/master/testing/python/demo.ipynb

[25] Z. Cao, T. Simon, S. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," *CoRR*, vol. abs/1611.08050, 2016. [Online]. Dosegljivo: http://arxiv.org/abs/1611.08050

[26] D. E. King. (2017) dlib c++ library. [Dostopano: 25.9.2017]. [Online]. Dosegljivo: https://github.com/davisking/dlib

[27] D. E. King. (2017) Dlib find candidate object locations. [Dostopano: 25.9.2017]. [Online]. Dosegljivo: https://github.com/davisking/dlib/blob/master/python_examples/find_candidate_object_locations.py

[28] Y. Uchida. (2017) Keras implementation of a cnn network for estimating age and gender. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: https://github.com/yu4u/age-gender-estimation

[29] A. Geitgey. (2017) The world's simplest facial recognition api for python and the command line. [Dostopano: 29.8.2017]. [Online]. Dosegljivo: https://github.com/ageitgey/face_recognition